Certificate of Mailing/Facsimile 37 CFR §1.8(a)

I hereby certify that this correspondence is being:

__X__ deposited with the United States Postal Service as first class mail in an envelope with sufficient postage addressed to the:

____ facsimile transmitted to **[Fax Number]** to the:

Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on July 5, 2005.

Susan L. Phelps

Signature

In the United States Patent and Trademark Office

Date: July 5, 2005

In re Application of: Moyer et al.

Filed: October 25, 2001

For: Critical Adapter Local Error Handling

Serial Number: 10/028,525

Conf. No.: 4463

Art Unit: 2113

Examiner: Joseph D. Manoskey

Declaration Under 37 CFR § 1.131

Hon. Commissioner for Patents
P.O. Box 1450, Alexandria, VA 22313-1450

Sir:

I, Dawn Moyer, declare the following to be true, to the best of my knowledge and recollection:

1. that I am one of the inventors of the subject invention disclosed and claimed in the above-identified patent application;

2. that I was employed by International Business Machines Corporation in New York at the time of the subject invention and have continued to be so employed to the present;

3. that the subject invention described in the above-mentioned application was conceived and reduced to practice in Poughkeepsie, New York, prior to July 17, 2001;

4. that the subject invention was embodied in code that was based upon an approved component design document which was also supplied to an independent (within the International Business Machines Corporation organization) Functional Verification Test (FVT) group whose responsibility was to exercise the functionality specified in the component design document;

5. that the subject code embodying the subject invention was actually approved by the Functional Verification Test group prior to July 17, 2001, and then was also subsequently passed on to a separate System Test group whose responsibility it was to insure compatibility within the larger system environment and with other running unrelated code packages, all of this occurring before July 17, 2001;

6. that the subject code embodying the subject invention was "closed" by the System Test group prior to July 17, 2001, meaning that the subject code had completed all of the testing phases required by the International Business Machines Corporation;

7. that, in accordance with International Business Machines Corporation software release procedures, code is not released for general availability prior to full and complete testing by the System Test group;

8. that code embodying the subject invention was announced as being "generally available" (in accordance with the same meaning given that phrase in recitation #7 immediately above) before the end of the year 2000;

9. that included herewith is a copy of the above-mentioned component software design document that was used to implement the features of the subject invention;

10. that all files, functions and their associated functionality recited in the claim of the present application were designed, implemented and operative to support Feature #47587 and are documented in the approved component design document; and that the component design document contains the ioclt names and not the actual function and file names, which are left up to the code developers. (For instance, **cadd_adapter_start** in file **start.c** is documented as **ioctl ADAPTER_START**. The **ioctls** are used by the FSD whereas direct function calls are used by the device driver. The **ioctl** is merely a wrapper for user space code to access the hardware but it calls the exact same functions.);

11. that, with respect to the first recited step of claim 1 (that is, "detecting a nonpermanent error condition, within an adapter connected to one of said nodes, from which recovery is possible from within the node connected to said adapter"), this step was implemented as follows: the detection of a potential error condition is identified via error class masks that correspond to each specific error interrupt register on the adapter. These masks classified which bits were treated as possible recoverable critical adapter errors. The FLIH (first level interrupt handler) and SLIH (second level interrupt handler) functions ***cadd_intr*** and ***cadd_intr_offlvl*** in file ***cadd_intr.c*** respectively on the affected node classified and responded to all hardware error interrupts generated by the adapter. Eventually this classification and HW error registers were passed along to the FSD (Fault Service daemon) for further actions;

12. that, with respect to the second recited step of claim 1 (that is, "suspending communications from within the node with the adapter affected by said error condition"), this step was implemented as follows: Communications are suspended from within the node via the suspension of all existing open windows and rejecting any new window opens on the adapter experiencing the critical adapter error condition. A CSS_SUSPEND_WINDOW event is posted

to all registered window owners (HAL and IP (Hardware Abstraction Layer and Interface Protocol)) which leave the windows open, drop packets and return successfully on reads and writes. There is no explicit notification to the protocols (APIs) that the window resources are no longer available. This notification caused the protocols to terminate the running applications 100% of the time. The FLIH function ***cadd_intr*** in ***file cadd_intr.c*** used the ***cadd_suspend_windows*** function in file ***cadd_intr.c*** to suspend communications within the node without termination of any running applications. HAL (Hardware Abstraction Layer) used function ***_col_suspend_win*** in file ***col.c*** and IP used function ***ifcl_suspend_window*** in file ***ifcl_cfg.c*** to take appropriate actions on the posted suspend event from the FLIH;

13. that, with respect to the third recited step of claim 1 (that is, "disabling communication between said affected adapter and said switch so as to provide an indication to at least one other node in said network that communication with said affected adapter is at least temporarily suspended so as to effectively cause suspension of, but not termination of, applications running on said at least one other node in said network "), this step was implemented as follows: The SLIH (Second Level Interrupt Handler) for critical adapter errors on the affected node resets the affected adapter to clear up any possible hang conditions. Resetting the adapter resulted in disabling the link (link no longer timed) which in turn caused link sync failures to be raised to the FSD switch recovery code running on the Primary node (FSD central point of control). Switch recovery suspended thresholding of link sync errors for 10 seconds. After 10 seconds, switch recovery continued to count link syncs within a specific time period for thresholding purposes. If the recovery actions on the affected node did not re-enable the link within the 10 second recovery window, the FSD switch recovery on the primary node "thresholded" and fenced the affected adapter off the switch. This fence (adapter recovery failure path) released all window resources on the affected adapter which resulted in the termination of running applications using these resources. The SLIH disabled communications between the affected adapter and the primary FSD node via the function ***cadd_adapter_reset*** in file ***reset.c***. The FSD switch recovery suspended thresholding and fenced the adapter upon meeting the link sync threshold via function

CS_Switch_error_ recovery in file ***CSrecovery.c***. Resources were released via the existing FSD base function ***cadd_adapter_resource_release*** in ***file cadd_auth.c.***

14. that, with respect to the fourth recited step of claim 1 (that is, "performing recovery operations, at said affected node, to restore operation of said affected adapter, based on said detected error condition, said recovery including enablement of said disabled communication"), this step was implemented as follows: The FSD adapter function ***fs_daemon_fsm_adapter_ thread_main*** in file ***fsd_fsm_adapt.c*** started the adapter which re-enabled the link via function ***cadd_adapter_start*** in file ***start.c***.

15. that, with respect to the fifth recited step of claim 1 (that is, "resuming communication with said affected adapter upon enablement of said disabled communication"), this step was implemented as follows: The FSD resumed communication on the affected adapter after successful recovery actions by resuming the suspended windows on the adapter via ***cadd_resume_windows*** in file ***cadd_auth.c.***

16. that the functions, calls, files and processes described above are found in the included component design document, subject to the naming clarification set forth in Item No. 10 above.

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Respectfully Submitted,

7/5/05
Date

Dawn S. Moyer
Dawn S. Moyer

Component Design
Colony Adapter Recovery

Document Owner
Dawn S. Moyer
email:dawnstar@pk705vma

Department 42JA
POWER Parallel Systems
Poughkeepsie, N.Y. 12601

Version 4.0
Monday, February 08, 1999

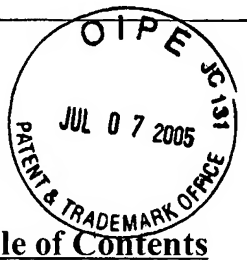


Table of Contents

Document Location

A postscript version of this document can be found in:

/afs/aix.kingston.ibm.com/depts/42ja/mohonk-colony/component_docs/switch/designs/col_recovery.ps

Design Review History

Version 1.0 - Electronic review May 22, 1998

Version 2.0 - Electronic review October 30, 1998

Version 3.0 - Sit down review January 7, 1999

Version 4.0 - Completed design distributed February 8, 1990

Reference Documents

1. **Scalable Parallel (SP) Colony Communications Adapter Functional Specification**
   - by Department G6Ka
2. **Mohonk Multi-threaded Switch Fault Service Daemon**
   - by R.F. Bartfai, C.H. Crawford, & J. Sayre
3. **Colony Adapter Kernel Extension & Device Driver**
   - by Detmetrios K. Michailaros, Janet Morgan, Dawn S. Moyer, Aruna Ramanan & Johannes Sayre

Acknowledgments

I would like to thank the members of the Colony Switch/Adapter and CSS development teams for their participation and contributions made during the design process. In particular I would like to thank Jonathon Kaufman, Nick Rash, Johannes Sayre, Jim Gregerson and Mike Grassi.

1.0 Design Overview

1.1 Terminology

The following phrases and terms are frequently used throughout this document:

- Reset the adapter:
  - Take the adapter off-line where the:
    - microcode is off-line and 6XX master traffic is quiesed.
    - slave 6XX traffic is quiesed (630 window owners) prior to resetting the adapter.
    - chips are reset but SRAM and RAMBUS are maintained (not reset).
- Initialize the adapter:
  - Brings the adapter back on-line after a reset. Adapter and microcode are operational.
- Refresh the adapter
  - Resets and initializes the adapter.
- Take the port off-line
  - TBIC is not synchronized with the switch and cannot send or receive data over the port.
- Permanent adapter errors
  - Unrecoverable adapter errors that reset the adapter and wait for administer intervention.
- Critical adapter errors
  - Recoverable adapter errors that suspend windows, allow no new windows, refresh the adapter and resume windows. An attempt is made to maintain running jobs and pending jobs are stalled through this recovery scenario. These errors may threshold to a permanent error.
- Permanent port errors
  - Port errors that take the port off-line. Administrator hardware intervention may be required to recover the port in error. Once this has been accomplished the software will reestablish the port in error.
- Transient errors
  - Adapter and port errors that are ignored but may threshold to a critical or permanent error.
- FLIH
  - System second level interrupt handler. Determines whether the associated adapter caused the interrupt and does its necessary work to schedules more off-level work and returns INTR_SUCC. Used in this document to define the device driver FLIH that invokes the off-level error SLIH.

1.2 Objectives

The main objectives of Colony adapter recovery are to:

- support the n ports per adapter and n adapters per machine model. currently n is equal to 2.
- attempt to maintain running and pending jobs on critical adapter errors. Previous adapter recovery designs disabled the ports (adapter and switch) and exited the daemon on these types of errors (called permanent errors) which then required administrative intervention to recover the adapter (e.g. rc.switch or reboot). Since some errors proved to be recoverable after a reboot or rc.switch (e.g.: data parity errors) and did not resurface for long periods of time it has become desirable to recover automatically without interrupting the running and pending jobs.
- threshold on all adapter and transient errors and make these thresholds tunable. Previous adapter recovery designs only thresholded on transient errors and maintained the same threshold for all (except bad packet).

1.3 Overview

Colony adapter recovery is the function that has responsibility for recognizing and taking action on all local adapter errors for the n port Colony adapter. This includes defining:

- local error interrupts and the corresponding actions required by both the RS6K device driver first level interrupt handler FLIH and off-level error SLIH .
- local error hardware classifications and the corresponding hardware actions required for resetting, reinitializing the adapter, taking the port off-line and managing windows.
- transient error to critical adapter or permanent port error thresholds.
- critical adapter error to permanent adapter error thresholds.
- the appropriate snaps at different error levels (node/adapter/port).
- local error message classifications for CSS error logging.
- notification events to the window owners (IP/HAL/KHAL) and the associated actions (protocol notification and window management).

1.3.1 Component Roles

There are five software components involved in Colony recovery. They are the RS6k device driver FLIH/ error SLIH and ioctls, the adapter microcode, the hardware abstraction layers (HAL/KHAL) and IP, the fault service daemon (FSD) and *Col_ recovery* (local error case from within the fault service daemon). *Col_recovery* will be discussed as a separate component for purposes of this document.

Note: In the future *Col_ recovery* may be moved into device driver code.

Adapter Microcode

The interrupt handler portion of the adapter microcode is responsible for bringing the adapter into suspend state and flushing it's cache to SRAM on permanent port errors. The microcode is also responsible for activating 6XX error interrupts on bad packets and service window full error conditions.

Device Driver

The FLIH/error SLIH are responsible for reading local error registers and putting corresponding requests onto the adapter work request queue (WRQ). They handlers are also responsible for quiescing slave traffic and attempting to reset the adapter on critical adapter errors.

The device driver also provides the necessary ioctls to bring the adapter down, initialize it back up again, manage the windows,notify the protocols...etc.

HAL/KHAL/IP

The window owners are responsible for responding to events to suspend or close their windows and notify the protocols.

Fault Service Daemon

The FSD is responsible for passing the local error request from the adapter WRQ to *Col_recovery* updated with the adapter name and fd.

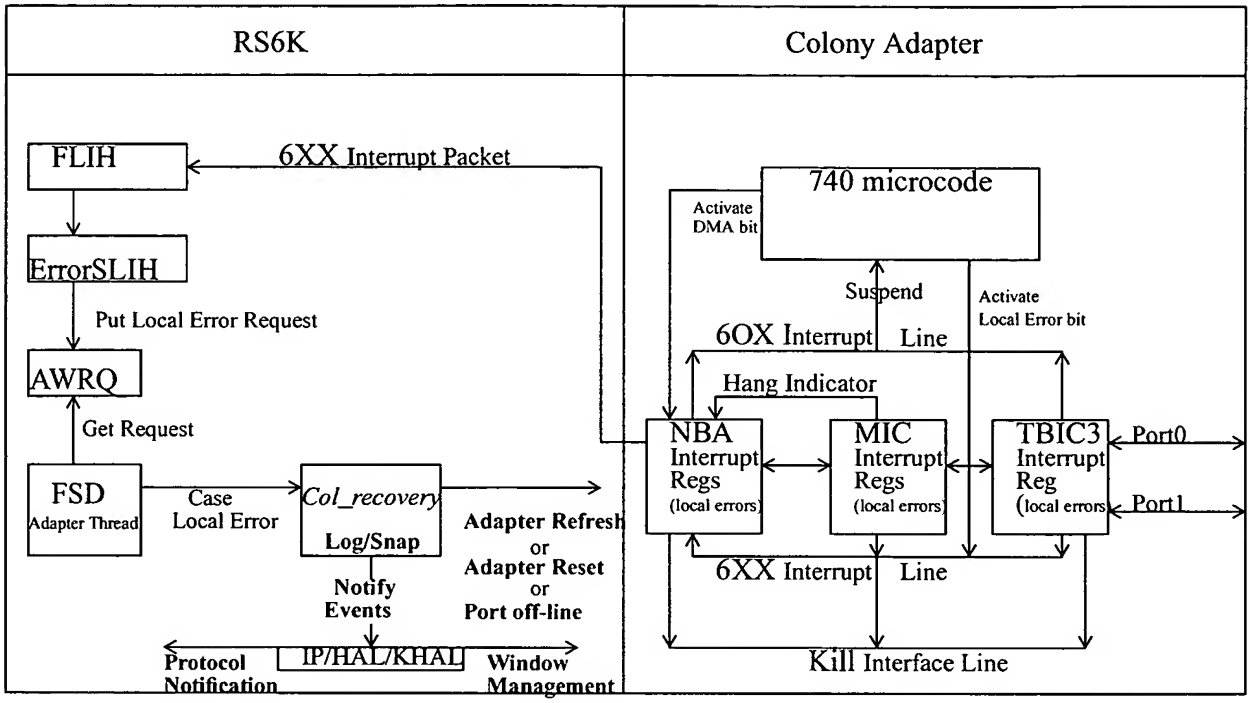
The FSD also provides adapter and port thread locks, WRQ, vFIFO and connectivity matrix software management routines.

Col_recovery

Col_recovery is responsible for classifying the local error and invoking the hardware actions necessary for taking the adapter or port off-line, monitoring and escalating critical and transient thresholds, notifying the window owners to suspend or close windows and notify protocols, logging errors and taking snaps.

1.3.2 Component Control Sequence

The following diagram shows the software components on the RS6K and the hardware and microcode on the Colony adapter:



The recovery sequence begins when

1. One of the colony adapter chips (NBA/MIC/TBIC3) experiences a local error and activates an interrupt register error bit, or an outside source activates the bit. This may active the 6XX interrupt, 6OX interrupt or NBA hang indicator bit and the kill interface lines depending how the bit was classified as follows:
   - all local errors will activate a 6XX interrupt.
   - all TBIC3 port permanent errors will also activate a 6OX interrupt causing the microcode to suspend.
   - all critical adapter errors will also activate the kill interface.
   - all MIC interchip bus hang errors that render the MIC, TBIC3 and SRAM unreadable and are 6OX interrupt enabled will activate the hang indicator bit in the NBA RHS error accumulator interrupt register versus the 6OX interrupt line.

Design Change: *The interrupt handler has been redesigned to reset the adapter in the event of a critical adapter error (MIC interchip bus hangs) before reading or writing the MIC, TBIC3 and SRAM. These errors are now defined to 6XX interrupt ONLY.*

Note: Please refer to Appendix A for bit specific classification and setup.

2. On a 6OX interrupt the microcode will go into a suspend state and wait to be resumed.
3. On a 6XX error interrupt the FLIH and off-level error SLIH.will :
   - stop global interrupts.
   - notify the window owners (HAL/KHAL/IP) to suspend windows on critical adapter errors or close windows on permanent adapter errors.
   - reset the adapter before reading the interrupt status and error registers on critical adapter errors.
   - reset interrupt status and error registers and disable associated interrupt enable registers.
   - disable the stop global interrupts on non critical adapter errors.
   - put the error request on the head of the AWRQ.
4. The FSD adapter thread will get the request from the adapter WRQ making it available to *Col_recovery.*
5. *Col_recovery* takes the following actions:
   - decode bits to determine error classification.
   - on permanent adapter errors not alraedy reset by the off-level error SLIH, reset the adapter and terminate the port and adapter threads.
   - on critical adapter errors, maintain threshold counts and refresh the adapter (maintaining running jobs).
   - on port permanent errors take the port off-line, interrupt the microcode to flush it's cache to SRAM and terminate the port thread.
   - on transient errors, maintain transient threshold counts and test for hot interrupts, escalating to a critical adapter or port permanent errors when either condition is met.
   - resume suspended windows for critical adapter errors or close windows on permanent adapter errors.
   - write CSS error log on all errors and snap on critical and permanent adapter errors.

2.0 Design Directions & Requirements on other Components

2.1 Adapter Microcode

The following sections provide specific details on adapter microcode requirements and it's interactions with other recovery components.

2.1.1 Handling and Raising Interrupts

Responsibilities for other components are listed for completeness.

- An interrupt to the microcode may be initiated in one of two ways:

- **hardware activated status thresholds (TBIC_ISR bits 11,12)**
  - no specific requirements.
- ***Col_recovery*/Error SLIH activated 6XX to 6OX intentional interrupt (NBA status bit 60)**
  - the microcode is expected to go into a suspend state at it's earliest possible convenience (if not already) and flush it's cache to SRAM. Used by the off-level error SLIH on a port permanent errors. Used by *Col_recovery* when transient port errors escalate to port permanent.

- A 6XX interrupt may be initiated from the microcode in one of two ways:

- **DMA Complete (NBA_ISR bit 11)**
  - no specific requirements.
- **Error Interrupt: Bad (data/svc) Packet or Service Window Full (NBA_ISR bit 61)**
  - the microcode is expected to issue a 6XX interrupt when a bad packet or the service window is full condition is detected just as it does today. The number of the port that experienced the error must also be included in the interrupt vector structure. All these errors are transient port errors that *Col_recovery* will threshold on and take the port off-line when the threshold has been met. The service window full condition will also dump the contents of the send and receive FIFOs through a snap.

Designer Note: *Col_recovery must interrogate the packet information to determine the port in error. A bad packet will always contain the correct port in error. However the service window full packet will merely contain the port of the packet incurring the full conditions which may or may not be the flooded port. Since both ports are thresholded on, it is likely that the flooded port will threshold first leaving the good port enabled.*

The microcode is expected to raise a waiting error interrupt before a pending DMA interrupt to avoid starvation. The microcode is also expected to wait for an ack back before issuing another interrupt (DMA or error).

Note: See the external_interrupt and error_interrupt pseudo code (off the Colony Microcode Design Page) for current design of microcode interrupt handler

2.1.2 Adapter Dump

The adapter dump will need a new parameter to differentiate between adapters. *Col_recovery* also requires the -n option to be maintained (assumes that the device driver or daemon has flushed the cache). Critical adapter errors reset the adapter which uses a soft reset to flush the cache and a hard reset to take the microcode off-line before snapping. *Col_recovery* will be responsible for checking the completion flag before snapping.

2.1.3 Suspend and Resume State

In suspend state, the microcode will process service packets, act on resume and load route commands, discard data packets on the receive side and stop draining the send FIFOs. The microcode will still raise error interrupts.The microcode will remain in suspend state until it receives a resume command or is taken off line with a hard reset. The microcode can also be put into the suspend state from a command interface. This command interface ADAPTER_SUSPEND ioctl is used by the FSD whenever route tables need to be loaded (e.g.: Fence/Start). *Col_recovery* uses the interrupt mechanism ADAPTER_HARD_SUSPEND ioctl instead of the command interface ioctl because:

- hardware interrupts are faster and faster is better when there is a bad port.

  Designer Note: *The kill interface cannot be used (versus suspend) here because the microcode would be unable to process the load route command.*

- intentional and hardware port permanent interrupts flush the processor cache to SRAM, which is needed for doing an adapter dump on a port permanent error. The command interface simply suspends without flushing the cache to SRAM.

The microcode is expected to return unsuccessfully when a resume adapter command has been received and the microcode has not been suspended. This eliminates the possibility that *Col_recovery* could resume the microcode prematurely leaving it in suspend state indefinitely.

2.1.4 Kill State

The kill interface is defined by *Col_recovery* to be hardware enabled on all critical adapter errors. The kill interface is also enabled directly from the ADAPTER_RESET ioctl. No actions are expected from the microcode. Activating the kill interface line forces the adapter into kill state, which stops DMA master operations and fences the ports (TBIC is not synchronized with the switch and cannot send or receive data over the ports). When the adapter is in kill state and the microcode puts a DMA request on the NBA command fifo no further microcode interrupts will be raised (error or DMA). This is due to the fact that NBA will never raise the interrupt before the DMA actually completes resulting in no acknowledgment to the microcode from the FLIH. However DMA requests already in progress will complete. Master operations are quiesced, slave operations and interrupts are not. The microcode must NEVER activate the kill interface.

Designer Note: *Although both sides may still read and write SRAM (microcode can respond to commands that do not require 6XX master operations) when the kill interface is active it is not recommended. We cannot guarantee under some critical adapter error conditions and during reset that a read to the adapter will return. If the processor times out on the read to the adapter the node will ckeckstop.*

2.1.5 Soft and Hard Reset State

The microcode is expected to flush the processor cache to SRAM and set a completion flag of 20 (in *g_window*) which ADAPTER_RESET ioctl uses to determine if it can dump the adapter after a reset has taken place. The microcode will then wait for a hard reset. The microcode is put into a soft reset state on critical and permanent adapter errors where *Col_recovery* does not want the microcode responding to commands. The adapter reset scenario within the off-level error SLIH or the ADAPTER_RESET ioctl will issue a hard reset following the soft reset to the microcode to take the microcode off-line but will not release the hard reset (bring back on line) before a snap is taken.

2.2 Device Driver

The device driver has two pieces, the interrupt handler FLIH and off-level error SLIH, and ioctls. The interrupt handler is responsible for processing DMA complete and local error interrupt requests. Ioctls provide the interface for *Col_recovery* to reset and restart the adapter and to quiesce master and slave traffic over the adapter.

2.2.1 Quiescing Slave Traffic on the Adapter

It is necessary to quiesce all slave traffic on the adapter prior to a reset or when a critical adapter error is active for the following reasons:

- there may be an undetectable bus hang present on the adapter.
- the adapter is in some stage of the reset state.

In either case some portions of the adapter will not be accessible. A read to these areas on the adapter will cause the processor to time-out and checkstop the node. Writes to the send command FIFOs in SRAM could also potentially backup internal buses rendering the adapter inaccessible.

The following design updates have been made to help quiesce slave traffic and avoid hang conditions:

- to quiesce the FSD slave traffic, all ioctls that access the adapter hardware must first check to see if the kill interface bit in RHS NBA (critical error indicator) is active and return an associated errno (TBD).
- to quiesce the window owners (HAL/KHAL/IP) from servicing us/ip data and service traffic, a new event and ioctl have been established. To quiesce the switch clock api clients from accessing adapter a new function has been established to notify all event threads. The FLIH will quiesce all slave traffic on critical adapter errors proir to resetting the adapter. However a window still exists between the time the critical adapter error actually occurs (kill interface active) and the FLIH gets control.
- to avoid possible hang conditions the off-level error SLIH will reset the adapter on critical adapter errors prior to reading any registers beyond the NBA. The reset will clear up any undetectable interchip and 6OX bus hang conditions that may have been present.

2.2.2 Interrupt Handling

The SLIH has the following tasks when processing local error requests.

- return an INTR_SUCC to the processor when:
    - the hard reset interlock cannot be obtained because the adapter is in the process of being reset through the ADAPTER_RESET ioctl.
    - the interrupt is for this device.

- critical adapter errors must:
  - supersede DMA completes because critical adapter errors may cause bus hangs which would hang the processor when reading the interrupt vector in SRAM. Overridden DMA completes are not regenerated because critical adapter errors stop global interrupts and reset the adapter.
  - notify all slave traffic to quiesce.
- stop global interrupts (COLA reset register bit 17) on all error interrupts. This is a necessity because the INTR_SUCC will be returned before the off-level error handler can disable interrupts. When the INTR_SUCC is returned to the processor an EOI (end of interrupt) is routed to the NBA mastering the interrupt. This EOI along with the active and interrupt enabled bit will cause the NBA to raise the same interrupt again.
- invoke the error handling path as an off-level interrupt handler. This is handled off-level because reads and writes to the adapter registers exceed the interrupt priority class level 2 maximum of 600 machine cycles. Machine cycle estimates were based on the following criteria supplied by Tom Dubois:
  - **NBA <-> 630:** 50 bus/**100 processor cycles**
  - **MIC <-> NBA:** 15 bus cycles; **MIC <-> 630:** 65 bus/**130 processor cycles**
  - **TBIC <-> NBA:** 27 bus cycles; **TBIC3 <-> 630:** 77 bus/**144 processor cycles**
  - **SRAM <-> 630:** 80 bus/**160 processor cycles**

  Designer Note: *Off-level error interrupt handler maximum must not exceed 5000 machine cycles. A maximum of 21 reads and writes at a maximum of 160 processor cycles each would yield 3360 processor cycles.*
- the off-level error handler will:
  - on all transient and port permanent errors:
    - reset all error and status interrupt registers as described under Appendix A.
    - disable all 6XX interrupt enable registers associated with active error bits except for microcode generated errors. *Col_recovery* will be responsible for re-enabling transient error interrupt bits that have not met their threshold. Thus the SLIH must obtain the ADAPTER_REGISTER_UPDATE equivalent lock before disabling any registers with transient errors. The SLIH will also be responsible for disabling 6XX interrupts on all port permanent and transient errors associated with a port permanent error (PP0: 35,36,40,41,46,47,50 PP1: 38,39,52,53,58,59,62).
    - disable stop global interrupts and ack back to microcode on microcode generated interrupts.
  - on critical adapter errors:
    - query all slave traffic to ensure that it has quiesced. If it has quiesced reset the adapter prior to reading any registers beyond the NBA to avoid potential bus hangs. If it has not look for interchip bus hang condition prior to reading any registers beyond the NBA.

- do not reset and disable interrupt enable registers. These errors will always do a global stop interrupt and an adapter refresh. No new interrupts will be generated and the registers will be reset and re-enabled through the refresh.
- drain the error pool of the AWRQ on critical adapter errors.

- on all errors update the FSD request element with:
  - all interrupt error and status interrupt registers when any local error has been detected. Return registers in an array ordered as follows:
    - NBA_IER, NBA_ISR, MIC_IER, MIC_ISR, TBIC3_IER
    - the ISRs/IERs must also be cleared off any detected error bits that are not enabled for interrupts.
  - the interrupt vector structure and the associated bad packet description return code and error data information.
  - a bit mask identifying registers where errors were found.
  - a snapshot of the MIC_ISR and TBIC3_IER after reset has been performed. Col_recovery uses the snapshot for hot interrupt testing of transient errors.
  - the interrupt type.
- put local error requests on the error pool of the adapter WRQ (fast request) when an error bit is active on one of the interrupt error or status registers and the associated 6XX enable bit is active. If the error request cannot be put on the queue the error request should be dropped.

2.2.2.1 Funtional Flow

This flow is intended to demonstrate requirements and NOT to dictate implementation. The interrupt handler (FLIH/SLIH) error paths combine with the *Col_recovery* error path to complete the entire main line recovery path.

BEGIN DD FLIH or System SLIH

```
interrupt_type = none; DMA_interrupt = F
/*-------------------------------------------------------------------------------------------------------------------------------------------------------------*/
/* No new interrupts will be generated from the NBA after a critical adapter or interchip bus hang error have been handled.                                   */
/*-------------------------------------------------------------------------------------------------------------------------------------------------------------*/
If (interrupt was not from this interrupt handler's device (eg:css0/css1))
   return INTR_FAIL /* Processor will give to next device */
/*-------------------------------------------------------------------------------------------------------------------------------------------------------------*/
/* If a reset is in progress (ADAPTER_RESET ioctl also acquires lock) we should return successfully. Since the reset will disable                             */
/* interrupts the NBA will not regenerate this interrupt. If another source has acquired the lock the interrupt will be regenerated.                          */
/*-------------------------------------------------------------------------------------------------------------------------------------------------------------*/
If (acquire the hard reset interlock) return INTR_SUCC
read NBA_IER
read NBA_ISR                                                    / * Note most of latency is buried by the read to NBA_ISR.*/
```

```
/*-------------------------------------------------------------------------------------------------------------------------------------------------*/
/*- If the KILL interface is active we MUST have a critical adapter error. ONLY critical adapter errors and the                              */
/*   ADAPTER_RESET ioctl activate the KILL interface and a reset would have kept us from getting the lock.                                      */
/*- Critical adapter errors MUST supersede DMA complete because we should not access SRAM. we cannot guarantee that the                      */
/*   critical adapter error has not caused a bus hang which will hang the processor when a read to the INTR VECTOR fails to                    */
/*   return. DMA will not be regenerated because CA errors stop interrupts and reset the adapter before we exit the error SLIH.               */
/* - Can be a microcode DMA or error (NEVER both).                                                                                             */
/*-------------------------------------------------------------------------------------------------------------------------------------------------*/
if (NBA_ISR ANDed NBA_ISR_KILLmask)
   interrupt_type = critical_adapter_error          /* Hardware generated error. Includes interchip bus hangs. */
   ADAPTER_SUSPEND_WINDOW ioctl equivalent                                  /* quiesce 6XX slave traffic HAL/IP */
   swclock_reset_handler(notify)                    /* quiesce 6XX slave traffic switch clock api clients */
endif
else
   if (NBA_ISR ANDed NBA_ISR_DMAmask)                    /* Microcode generated DMA. Never disabled. */
      read INT_VECTOR from SRAM global window            /* Multiple DMAs (window thresholds may be set)*/
      call appropriate DMA-SLIHs (HAL/KHAL/IP) /* generate events */
      /* reset active DMA complete and expected status bits NOT active error bits */
      write back NBA_ISR reg with ( NBA_ISR ANDed NBA_ISR_DMASTATmask )
      release the hard reset interlock
      DMA_interrupt = T
   endif
   if (interrupt_type == "") || (NBA_IER ANDed NBA_IER_HWerrormask)
      interrupt_type = trans_or_port_error            /* microcode or hardware generated error.   */
endelse
If (interrupt_type == trans_or_port_error || == critical_adapter_error)          /* error exists */
   /*----------------------------------------------------------------------------------------------------------------------------------------*/
   /* MUST stop the NBA from mastering new interrupts before they can be disabled. Does not disable interrupt enable register.   */
   /* Don't want microcode to raise another interrupt when a possible port error is present.                                      */
   /*----------------------------------------------------------------------------------------------------------------------------------------*/
   write NBA_COLA reset reg to activate GLOBAL 6XX STOP INTERRUPTS bit
   Call the system to schedule off-level error handler(NBA_IER, NBA_ISR, i interrupt_type)
endif
if (DMA_interrupt == T)
   ack back to microcode (write SRAM) so new interrupt can be generated (DMA/Error)
return INTR_SUCC
```

END DD FLIH or System SLIH

BEGIN DD OFF-LEVEL ERROR SLIH

```
Error_interrupt = F; microcode_error_interrupt = F
select interrupt_type
      case (critical_adapter_error)
         /*-------------------------------------------------------------------------------------------------------------------------------*/
         /* Col_recovery will resume suspended windows. If we can't put on the AWRQ the DD will release resources.                       */
         /*-------------------------------------------------------------------------------------------------------------------------------*/
         do until (errno!=EBUSY || timer_pop) /* Maybe 50 bus cycles */
            ioctl_rc1=ADAPTER_WINDOW_SUSPEND_COMPLETE(cssfd) equivalent
            ioctl_rc2=switch_reset_handler(complete)
         enddo
```

```
    if !(ioctl_rc1) || !(ioctl_rc2)                          /* slave traffic has quiesced */
       toggle COLA reset reg: activate 740 SOFT RESET bit  /* MC dump SRAM wait hard reset */
       write COLA reset reg: activate 740 HARD RESET bit /* Take MC off-line */
       sync; write COLA reset reg: reset RHS_NBA/MIC/TBIC3/LHS_NBA enable /* reset chips */
       sync; write to dummy pinned area; sync; wait 200 bus clocks; sync
       write COLA reset reg: disable reset RHS_NBA/MIC/TBIC3/LHS_NBA enable/*out reset */
       fs_request.reset = T
    endif
    else if (NBA_IER ANDed NBA_IER_IB_HANGmask) /* Cannot read MIC/TBIC regs or SRAM */
       interrupt_type = interchip_bus_hang
case (trans_or_port_error) /* may have both a microcode and hardware generated error */
    /*-------------------------------------------------------------------------------------------------------------------*/
    /* Since critical adapter and interchip bus hang errors never disable the global stop interrupts (allowing interrupts to */
    /* be mastered again) we need not disable interrupt enable registers or clear IER/ISR bits. Both of these things */
    /* will be accomplished through step 14 of the ADAPTER_START ioctl at refresh time. */
    /*-------------------------------------------------------------------------------------------------------------------*/
    fs_request.ErrReg[0] = NBA_IER ANDed NBA_IER_IB_HANGmask
    read NBA_6XX_ISMR
    fs_request.ErrReg[1] = NBA_ISR ANDed NBA_6XX_ISMR
    if (interrupt_type == interchip_bus_hang)          /* Cannot read MIC/TBIC regs or SRAM */
       fs_request.ErrReg[2/3/4] = NULL
       break
    endif
    if (fs_request.ErrReg[1])
       Error_interrupt = Error_interrupt ANDed NBA_ISR_ACTIVEmask
       if (interrupt_type != critical_adapter)
          if (NBA_ISR ANDed NBA_ISR_740mask)
             interrupt_type = microcode_trans_error
             read INT_VECTOR from SRAM global window /* svc window full or bad packet NOT both */
             fs_request.intVect = Interrupt Vector
             fs_request.err_rc = Bad Packet Description
             fs_request.err_data = Error data pertaining to bad packet description
             microcode_error_interrupt = T
          endif
          /* disable interrupts for active and 6XX enabled ISR bits except for 740 INTR (may come in HOT) */
          write back NBA_6XX_ISMR with ~( ( NBA_ISR ANDed NBA_6XX_ISMR )
                                          XORed NBA_ISR_740mask )
          /* reset active and 6XX enabled ISR bits.*/
          write back NBA_ISR with ( NBA_ISR ANDed NBA_6XX_ISMR )
       endif
    endif
    read MIC_ISR
    read MIC_6XX_ISMR
    fs_request.ErrReg[2] = MIC_ISR ANDed MIC_6XX_ISMR
    if (fs_request.ErrReg[2])
       Error_interrupt = Error_interrupt ANDed MIC_ISR_ACTIVEmask
       if (interrupt_type != critical_adapter)
          /* Must obtain new lock e.g. ADAPTER_REGISTER_UPDATE for the write to IEMR and IER. */
          /* reset active and 6XX enables ISR bits. Must obtain new lock e.g. ADAPTER_REGISTER_UPDATE */
          write back MIC_ISR with ( MIC_ISR ANDed MIC_6XX_ISMR )
          read MIC_ISR /* read the MIC ISR again for Hot interrupt testing.
          fs_request.MIC_ISRsnap = MIC_ISR
       endif
    endif
```

```
            read MIC_IER
            read MIC_6XX_IEMR
            fs_request.ErrReg[3] = MIC_IER ANDed MIC_6XX_IEMR
            if (fs_request.ErrReg[3])
               Error_interrupt = Error_interrupt ANDed MIC_IER_ACTIVEmask
               if (interrupt_type != critical_adapter)
                  /* reset active and 6XX enables IER bits */
                  write back MIC_IER with ( MIC_IER ANDed MIC_6XX_IEMR )
               endif
            endif
            if (MIC_IER ANDed MIC_6XX_IEMR ANDed MIC_IER_TBIC3_HANG(bits 10,19))/* !access TBIC */
               interrupt_type = interchip_bus_hang
               fs_request.ErrReg[4] = NULL
               break
            endif
            read TBIC3_IER
            read TBIC3_6XX_IEMR
            fs_request.ErrReg[4] = TBIC3_IER ANDed TBIC3_6XX_IEMR
            if (fs_request.ErrReg[4])
               Error_interrupt = Error_interrupt ANDed TBIC3_IER_ACTIVEmask
               if (interrupt_type != critical_adapter)
                  /* Must obtain new lock e.g. ADAPTER_REGISTER_UPDATE for the write to IEMR and IER.   */
                  /* If port permanent error disable interrupts for all port error bits associated with the port in error.   */
                  if TBIC3_IER ANDed (TBIC3_IER_PP0mask || TBIC3_IER_PP1mask)
                     write back TBIC3_6XX_IEMR with ( ~( TBIC3_IER ANDed TBIC3_6XX_IEMR )
                                                ORed TBIC3_IER_PP#TP#mask )
                     write TBIC CTRL reg 0 to disable port in error????
                  endif
                  if (TBIC3_IER ANDed TBIC3_6XX_IEMR ANded TBIC3_IER_TRANSmask)
                     write back TBIC3_6XX_IEMR with ~( TBIC3_IER ANDed TBIC3_6XX_IEMR )
                  /* reset active & 6XX enables IER bits but NOT shared 60X IER bits (microcode needs to see INTR too.   */
                  write back TBIC3_IER with (TBIC3_IER ANDed (TBIC3_6XX_IEMR XORed TBIC3_60Xmask))
                  read TBIC3_IER /* read the TBIC3 IER again for Hot interrupt testing    */
                  fs_request.TBIC3_IERsnap = TBIC3_IER
               endif
            endif
endselect
/*-------------------------------------------------------------------------------------------------------------------------------------------------------------------*/
/* Transient and port permanent errors have reset and disabled active error bits and MUST now allow the NBA to master new   */
/* interrupts (DMA and other errors). Critical adapter and interchip bus hangs do not want any new interrupts to be generated.   */
/*-------------------------------------------------------------------------------------------------------------------------------------------------------------------*/
if (interrupt_type != critical_adapter) || (interrupt_type != interchip_bus_hang)
   write NBA_COLA reset reg to disable GLOBAL 6XX STOP INTERRUPTS bit
if (microcode_error_interrupt == T)
   ack back to microcode (write SRAM) so new interrupt can be generated (DMA or error)
if (interrupt_type == critical_adapter)
   drain the error pool AWRQ
fs_request.ActiveErr_mask = Error_interrupt
release the hard reset interlock
```

```
if(Error_interrupt)
   call fault_service_inotice(fs_request) /* put request on error pool fast path to head of AWRQ */
      /* Adapter thread may be hung */
      do until (put request.ERROR on AWRQ(at head fast request)) || (timer_pop)
      if !(put request.ERROR on AWRQ)
         drop error request
   endcall /* fault_service_inotice*/
endif
```

END DD OFF-LEVEL ERROR SLIH

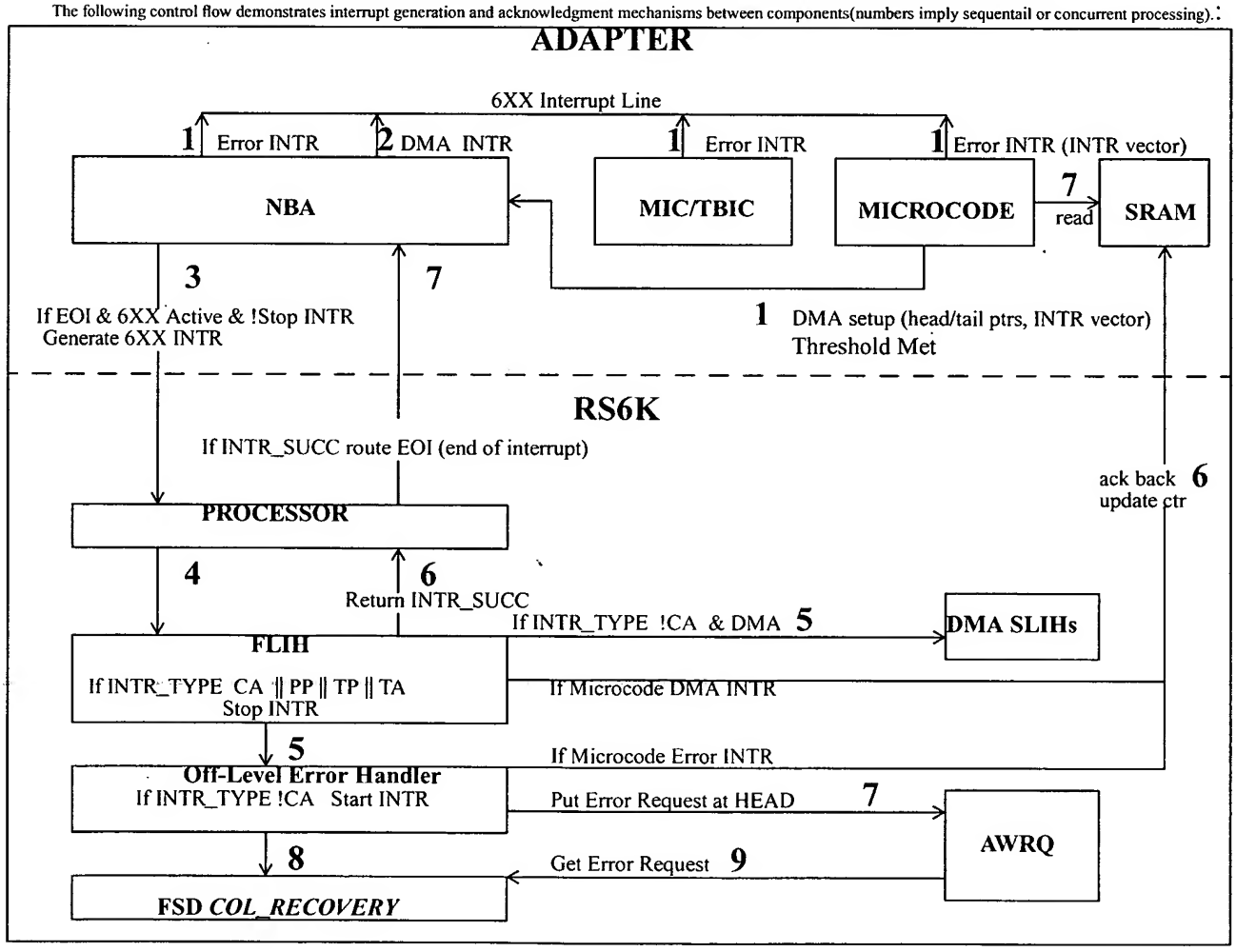
Note: Refer to Appendix E for the original/alternative interrupt handler design.

2.2.2.2 Component Control Flow

The interrupt handler design was based on the following assumptions:

- The NBA will master an interrupt when the 6XX line is active and an EOI has been received Multiple interrupts may be active when the 6XX interrupt is generated (e.g. Error Interrupt and DMA complete). If the global stop interrupt has been enabled from the COLA reset register no interrupts will be mastered until the bit has been disabled. Since the EOI is returned while the off-level error handler is in progress, interrupts must be stopped to ensure that the same interrupt is not raised again before the handler can interrupt disable the active error bits.
- The microcode will only generate one interrupt at a time (DMA || Error) waiting for an acknowledgment before generating another interrupt. If both a DMA and error interrupt are pending the error interrupt will be generated first. Error interrupts are generated by directly activating the 6XX interrupt line. DMA completes are generated by setting up a DMA request in the NBA CNTRL RESQEST FIFO. The NBA will activate the 6XX interrupt line once the DMA has completed successfully. The microcode will wait for an acknowledgment regardless of the outcome.
- When the kill interface is active (critical adapter errors) DMA is stopped. If an interrupt is generated with both a microcode (DMA/svc window full/bad packet) and a chip critical adapter error only the critical adapter error will be handled. Since critical adapter errors may cause a 6OX bus hang this is essential (read to SRAM would hang the processor).

The following control flow demonstrates interrupt generation and acknowledgment mechanisms between components(numbers imply sequentail or concurrent processing).:



2.2.3 I/O Controls New or Modified

2.2.3.1 ADAPTER_RESET

Designer Note: *All 630 slave traffic MUST be quiesced before this ioctl is issued. Since this may or may not include do_commands to the microcode and specific actions based on protocol response, this action should not take place in this ioctl. Interrupts, ports, 6XX and 740 master operations are quiesced through this ioctl, but all slave traffic must also be quiesced through the protocols (IP/HAL/KHAL window owners).*

- Set ioctl access to no-access and hard reset interlock respectively:
  - Prevents all ioctl access.
  - Prevents the SLIH from accessing the hardware.

Designer Note: *Need to do before kill interface is activated (must be able to determine source)*

- Enable NBA RHS ISR (bit 52) to **activate kill interface line**:
  - Shutdown or fence adapter interfaces. quiesces 740 master traffic and disables the ports. Does not quiesce 630 slave traffic. Keeps the master operations already on the NBA command fifo from being processed. This register MUST then be read back to ensure that master traffic is quiesced before resetting the chips.

  Note: For more information on the kill interface please refer to the kill state section under adapter microcode.
- Enable NBA LHS COLA toggle(active/inactive) register (bit 4) to **soft reset 740**:
  - Microcode is taken off-line until bit becomes inactive. 740 will then start executing from address 100 and flush processor cache to SRAM and set a completion flag of 20 which the *Col_recovery* uses to determine when it can dump the adapter. The 740 will then wait for a hard reset.
- Enable LHS COLA reset register (bit 3) to **hard reset 740**:
  - Takes the 740 off line until bit becomes inactive (ADAPTER_START ioctl). The 740 must be reloaded from SRAM for all recovery resets, because we cannot assume uncorrupted SRAM on a critical adapter error.
- Enable on NBA LHS COLA reset register (bits 0-2 and 9) to **reset RHS NBA/MIC/TBIC3/ LHS NBA enable**:
  - Disables all IER/ISR associated enable registers (6OX/6XX/KILL), but not the error or status registers themselves. This quiesces all RHS NBA/MIC/TBIC interrupts. Error and status bits must be reset through the ADAPTER_START ioctl.
  - Enables the reset lines forcing portions of the chips to a known state. Since the NBA RHS is in reset the chip cannot respond to any operations bound for the adapter (slave operations). Thus the COLA reset register should be disabled before exiting the ioctl so that snaps can be performed.
  - Resets all NBA LHS/RHS communications ptrs.

Note: Pass 2 addition to eliminate parking problem (expected mid September).

Update: Fix escalated new problem which renders this bit unusable. New RIT TBD.

- Disables associated enables in the Checkstop 6XX Event Tree registers, but not the accumulator bits themselves. Error accumulator bits must the cleared through the ADAPTER_START ioctl.
- Stops all LHS 6XX interrupts, checkstops and attentions.

Note: Pass 2 addition to eliminate parking problem did not encompass all of this bits functionality and problem stemming from fix does not effect enabling of this bit.

- Write to a pinned area and Wait 200 bus clocks:
  - The reset signals need to be active for a number of clock cycles and then the chips should also be allowed to quiesce for an additional number of cycles.
- Disable NBA LHS COLA reset register (bits 0-2 and 9) to **disable ASIC reset lines**:
  - Take the chips out of reset. The RHS NBA may now respond to slave operations (log and snap).
- Reset status/error interrupt registers and load control registers with appropriate quiesce bits.
- Release hard reset interlock and set ioctl lock access to reset-level:
  - ONLY the following ioctls may be issued:
    - READ_*/WRITE_*
    - *_ATTACH
    - QUERY_ADAPTER
    - ADAPTER_START

Designer Note: *We must maintain SRAM and RAMBUS for Col_recovery snaps and logging. Since the microcode load and MIC soft reset both take place in the ADAPTER_START ioctl, SRAM and RAMBUS are both maintained.*

2.2.3.2 ADAPTER_START

- Enable LHS COLA reset register (bit 12) to **soft reset MIC**:
  - This is a memory reset as described in the MIC Spec under resets (N0_M0_MEM_RESET). This step is normally done by the service processor at power on, but is also needed for recovery and daemon restart to flag that adapter facilities must be initialized (including RAMBUS and SRAM).
- Steps 13-18 in Greg Salyor's Service Processor document.
  - Step 14 resetting the status and error registers must:
    - reset the registers from port to 6XX bus order (TBIC/MIC/RHS & LHS NBA) because there are some registers that can hold another error/status register high if not disabled first. Must use bit 10 in the COLA reset register to reset the NBA IER.

- Step 18 must initialize:
  - all enable registers associated with error and status interrupt registers.
- Set the ioctl access lock to unlimited-access.

2.2.3.3 ADAPTER_HARD_SUSPEND

- Enable NBA RHS ISR (bit 60) to issue a **6OX interrupt to suspend the microcode**:
  - Causes microcode to suspend and flush cache. The microcode will process service packets, act on resume and load route commands, discard data packets on the receive side and fill up the send FIFOs. The microcode will remain in suspend state until it receives a resume command or is taken off line with a hard reset.

2.2.3.4 ADAPTER_RESUME

- Issue a command to the **microcode to resume normal operation**.
  - Microcode resumes normal operation after a ADAPTER_HARD_SUSPEND or ADAPTER_SUSPEND has been issued.
- Return unsuccessfully when microcode is not in suspend state.

2.2.3.5 ADAPTER_SUSPEND_WINDOW

- Post a CSS_SUSPEND_WINDOW event to all registered window owners:
  - The registered window owner routines (IP, HAL/KHAL(1 per job process)) are expected to quiesce all window traffic, mark the local window state as suspended and leave the window open. The window owners then will confirm the window suspension completion by invoking the ADAPTER_SUSPEND_WINDOW_COMPLETE ioctl. There will be no explicit notification to the protocols and all SRAM traffic (read/writes) will be quiesced (e.g. HAL_NOTIFY and HAL_WRITE_PKT requests). HAL and IP will simply drop the packet and return successfully on writes to the send command FIFO from ip and us clients (ip will ONLY discontinue writes after the current burst of 8 packets has completed). HAL will drop the packet and return unsuccessfully to the service client. Userspace and ip clients will eventually time-out if no ack is received back before the window is resumed. Requests to open new windows will be rejected by the kernel extension until the CSS_RESUME_WINDOW ioctl has been issued to reopen windows and resume normal operations. In the suspend window state the registered routines must also be ableto handle a CSS_WINDOW_CLOSE event where window owners will notify the protocols and invoke registered error handlers. This will eventually result in the termination of currently running jobs.
- Return success unconditionally

Designer Note: *Used on critical adapter errors where window traffic must be quiesced prior to resetting the adapter.*

2.2.3.6 ADAPTER_SUSPEND_WINDOW_COMPLETE

- Update window state to mark that the window is in suspend state. The registered window owner routines (IP/HAL/KHAL) are expected to invoke this ioctl at the completion of processing a CSS_SUSPEND_WINDOW event. Kernel extension will change the window state from started to suspended.

2.2.3.7 QUERY_SUSPEND_WINDOW_COMPLETE

- Gathers all currently opened window states.
    - return unsuccessfully when the all window are not in suspend state.

2.2.3.8 ADAPTER_RESUME_WINDOW

- Post a CSS_RESUME_WINDOW event to all registered window owners:
    - The registered routines (IP/HAL/KHAL) are expected to restart the window previously suspended through the CSS_SUSPEND_WINDOW event.

2.2.3.9 ADAPTER_RESOUCE_RELEASE

An additional parameter "adapter_access" will be added to this ioctl. The behavior of the ioctl will be based on the setting of the new third parameter as follows:

- adapter_access=yes
    - Post a CSS_WINDOW_DOWN event to all registered window owners:
    - The registered window owner routines (IP/HAL/KHAL) are expected to close all windows. If the windows have not been closed after some number of seconds the kernel extension will close the window and post a CSS_WINDOW_CLOSE to all registered routines. Registered routines will notify the protocols and invoke registered error handlers. This will eventually result in the termination of currently running jobs.
- adapter_access=no
    - Post a CSS_SUSPEND_WINDOW event to all registered window owners for all windows in started state (try to suspend all windows if not otherwise).
    - Change the state of the windows from started or suspended to opened. The kernel extension will close the window without doing a window close to the microcode and post a CSS_WINDOW_CLOSE to all registered routines. Registered routines will notify the protocols, invoke registered error handlers. This will eventually result in the termination of currently running jobs.

Designer Note: *The adapter microcode may not be accessible when Col_recovery invokes this ioctl with adapter_access=no. Thus we don't want to read SRAM through the do_command to close windows.*

2.2.3.10 PORT_DISABLE

- Disable Control Register 0 bits (20,59) to **disable TBIC ports**:
  - Disables ports #0 and #1 respectively.
- Monitor TBIC Status Register (bits 1 and 7 disabled):
  - Determines that ports #0 and #1 respectively are not functional.

2.2.3.11 READ/WRITE_NBA/MIC/TBIC/SRAM

- Write to the device driver message buffer when reading or writing the adapter. It is possible that we may encounter an unexpected interchip hang or 6OX bus hang (before interrupt handler can quiesce the 6XX slave operations). When this condition occurs the cadd dump will be the only source of problem determination.

2.2.3.12 ADAPTER_RECOVERY_INPROGRESS

- Read NBA RHS ISR register (bit 52) to **determine if the kill interface is active**.
- Return 0 unsuccessfully when not active.

Designer Note: *This ioctl is a good general indicator and will probably be needed by the FSD to determine whether a particular action should be taken when a critical adapter error has occurred while the current request type is being processed.*

2.2.3.13 ADAPTER_REGISTER_UPDATE

- Used by *Col_recovery* to re-enable transient hardware errors that have not escalated and to reset error interrupt registers during hot interrupt testing. This ioctl will obtain the write lock, perform the update operation and release the write lock before returning.

2.2.3.14 All I/O Controls that access Adapter Hardware

- A new requirement has been placed on all ioctls that access the adapter hardware.These ioctls must first check to ensure that the kill bit in the RHS NBA is NOT ctive. If the bit is active the hardware action will not be performed and an appropriate errno (TBD) will be returned indicating that adapter recovery is in progress. Since the kill interface is hardware activated at the time of the local adapter error, no windows exist.

2.2.4 Service & Userspace FIFO Dump

The new fifo dump will need a parameter to differentiate between adapters and will be expected to dump US (userspace), and service FIFOs.The is necessary because *Col_recovery* cannot determine which port the service window full error has been raised.

2.3 HAL/KHAL/IP

2.3.1 New Events for Critical Adapter Errors

On critical adapter errors *Col_recovery* needs to maintain the running job and quiesce 6XX slave traffic (don't access SRAM and RAMBUS).

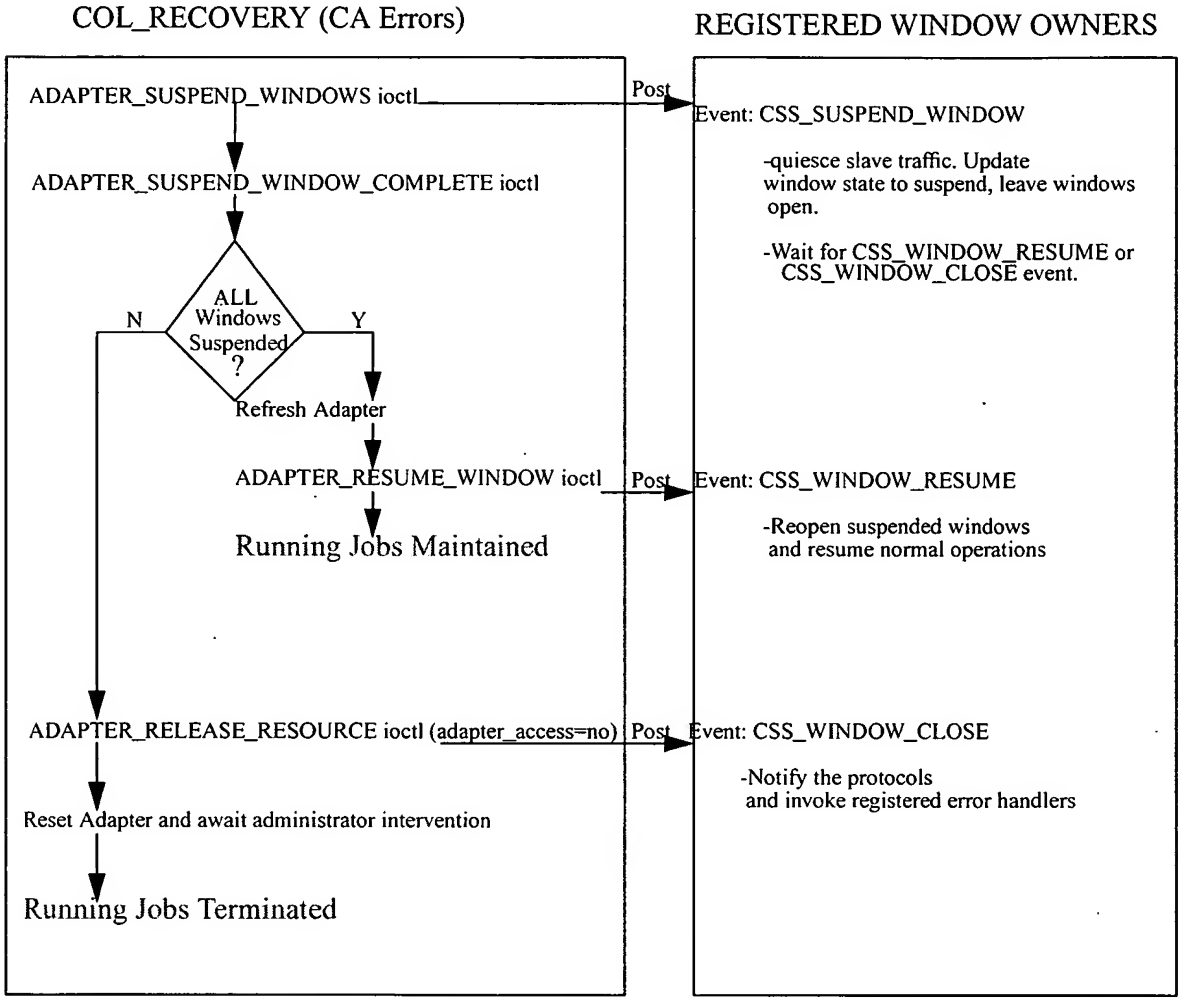
The registered window owners are expected to do the following:

- register for CSS_SUSPEND_WINDOW events and respond as follows:
    - update a variable through the ADAPTER_SUSPEND_WINDOW_COMPLETE ioctl confirming that the window state has been changed to suspend and leave the window open.
    - quiesce all SRAM read/write traffic (e.g. HAL_NOTIFY and HAL_WRITE_PKT requests). There should be no explicit notification to the protocols. HAL and IP should simply drop the packet and return successfully on writes to the send command FIFO from ip and us clients (ip will ONLY discontinue writes after the current burst of 8 packets has completed). HAL should drop the packet and return unsuccessfully to the service client. Userspace and ip clients will eventually timeout if no ack is received back before the window is resumed.
    - wait for CSS_RESUME_WINDOW or CSS_WINDOW_CLOSE event. Requests to open new windows will be rejected by the kernel extension until the CSS_RESUME_WINDOW ioctl has been issued to reopen windows and resume normal operations.
        - on a CSS_RESUME_WINDOW event:
            - restart the suspended window and resume normal operations.
        - on a CSS_WINDOW_CLOSE event:
            - notify protocols and invoke registered error handlers. This will eventually result in the termination of currently running jobs. *Col_recovery* will issue this event through the ADAPTER_RESOUCE_RELEASE ioctl when all of the registered window owners have not confirmed the suspend window state after receiving a CSS_SUSPEND_WINDOW event. The kernel extension will close the window without doing a window close to the microcode (adapter_access=no)

Designer Note: *Since critical adapter errors refresh the adapter (which includes reloading the microcode) all send FIFO and their contents will be lost.*

2.3.1.1 Component Control Flow

The following control flow demonstrates the event interactions and functionality between *Col_recovery* and the registered window owners.



2.3.2 IP Dump

The IP dump will need a new parameter to differentiate between adapters.

2.4 Threaded Fault Service Daemon

The FSD is expected to:

- update the local error request structure fs_req_int_data_t with adapter name and fd before passing control to *Col_recovery*:
- never activate the kill interface. The kill interface should ONLY be activated through critical adapter errors which are setup through the enable registers or through the ADAPTER_RESET ioctl.
- re-enable all port permanent and the corresponding transient errors associated with a port brought back on-line.
- disable the switch receive port and re-enable it back after 3 seconds when switch recovery detects a receive or send link error (autojoin too slow). If the error reoccurs within it's error interval the switch receive port will only be disabled. This behavior is necessary on critical errors where the ports are disabled, enabled and timed.

  Designer Note: *The LOAD_ROUTES ioctl which does require 6XX master operations cannot be performed when the KILL interface is active. Thus the switch recovery code that invokes the FSD function to load the route tables and do a cm_update should not be performed.*
- A new requirement has been placed on all ioctls that access the adapter hardware.These ioctls must first check to ensure that the kill bit in the RHS NBA is NOT active. If the bit is active the hardware action will not be performed and an appropriate errno (TBD) will be returned indicating that adapter recovery is in progress. The FSD will need identify all the paths that invoke these ioctls and evaluate the appropriate actions to be taken. Since the kill interface is hardware activated at the time of the local adapter error, no windows exist.
- A new ioctl ADAPTER_RECOVERY_INPROGRESS will be made available. This ioctl should be utilized by the FSD to determine the proper actions to be taken while adapter recovery is in progress and ioctls (as described above) cannot be used as indicators. Cases include exiting of port threads...etc. Since the kill interface is hardware activated at the time of the local adapter error no windows exist.

3.0 Col_recovery High Level Design

3.1 Overview

Col_recovery is responsible for classifying the local error and invoking all hardware actions necessary for taking the adapter or port off-line, monitoring and escalating critical and transient thresholds, quiescing 6XX slave traffic, logging errors and taking snaps. The device driver FLIH and off-level error SLIH share in some of these responsibilities and combined with *Col_recovery* complete the main Colony adapter recovery path. The following sections describe the *Col_recovery* path and it's relationship to device driver FLIH and off-level error SLIH.

3.2 Register Lookup Tables

Col_recovery will maintain a separate lookup table structure Adapter_ErrReg for each interrupt status and error register. The structure will contain the error type, threshold, and error message information pertaining to each bit along with adapter specific error masks...etc. The Adapter_ErrReg structure will become a member of the adapter_info structure located in the FSD fsd_multi_thd.h include file.

```
typedef struct {
     .
     .
     struct Adapter_ErrReg *aregs;
} adapter_info;

struct {
   char* reg_name;                                  /* specific error register name */

   enum classes { PA CA PP0 PP1 TA TP0 TP1 NONE }
   enum classes final_error_class = NONE,   /* returned final error type after decoding all bits */
   int final_error_bit;                             /* returned final error bit after decoding all bits */
   long  reg_active_mask,                    /* bit active in mask defining specific error register */
        /* All mask may be have bits ++ or -- from by escalated transient errors for search. */
        CA_mask,   initial_CA_mask,         /* mask defining CA error bits for error register */
        TA_mask,   initial_TA_mask,         /* mask defining TA error bits for error register */
        PP0_mask,  initial_PP0_mask,        /* mask defining PP0 error bits for error register */
        PP1_mask,  initial_PP1_mask,        /* mask defining PP1 errors bits for error register */
        TP0_mask,  initial_TP0_mask,        /* mask defining TP0 errors bits for error register */
        TP1_mask,  initial_TP1_mask,;       /* mask defining TP1 errors bits for error register */

   Boolean PP0_offline=F,PP1_offline=F; /* Port status.ONLY valid on TBIC3 register */

   struct {
      long  bit_active_mask,                 /* bit active in mask defining specific error bit */

      int threshold_cnt,                     /* current threshold count on bit. */

         threshold,                          /* current threshold on bit. TA only dual threshold.*/
         init_threshold,
         escalated_threshold,

         threshold_interval,              /* current time interval on bit. TA only dual threshold*/
         init_threshold_interval,
         escalated_threshold_interval;
```

```
        int     error_msg_num;          /* message number on bit */
        char*   error_cause,            /* error cause on bit */
                error_msg,              /* error message on bit */
                error_label;            /* error label (source of error) on bit */
        long    error_mask;             /* mask defining this bit */
    } error_bit[64]
} Adapter_ErrReg[NUM_ERR_REGS] { /* Initialize with APPENDIX A information. */
    { "NBA_ISR",,,,,,,{CA,,2,24hr,280,"DMA boundary error","A DMA request crossed the 4k
      boundary","COL_ADAPT_MC_ER",},
                        {,,,,,,,,} },
    { "NBA_IER",,,,,,,{,,,,,,,,},
                        {,,,,,,,,} },
    { "MIC_ISR",,,,,,,{,,,,,,,,},
                        {,,,,,,,,} },
    { "MIC_IER",,,,,,,{,,,,,,,,},
                        {,,,,,,,,} },
    { "TBIC3_IER,,,,,,,{,,,,,,,,},
                         {,,,,,,,;} }
};
```

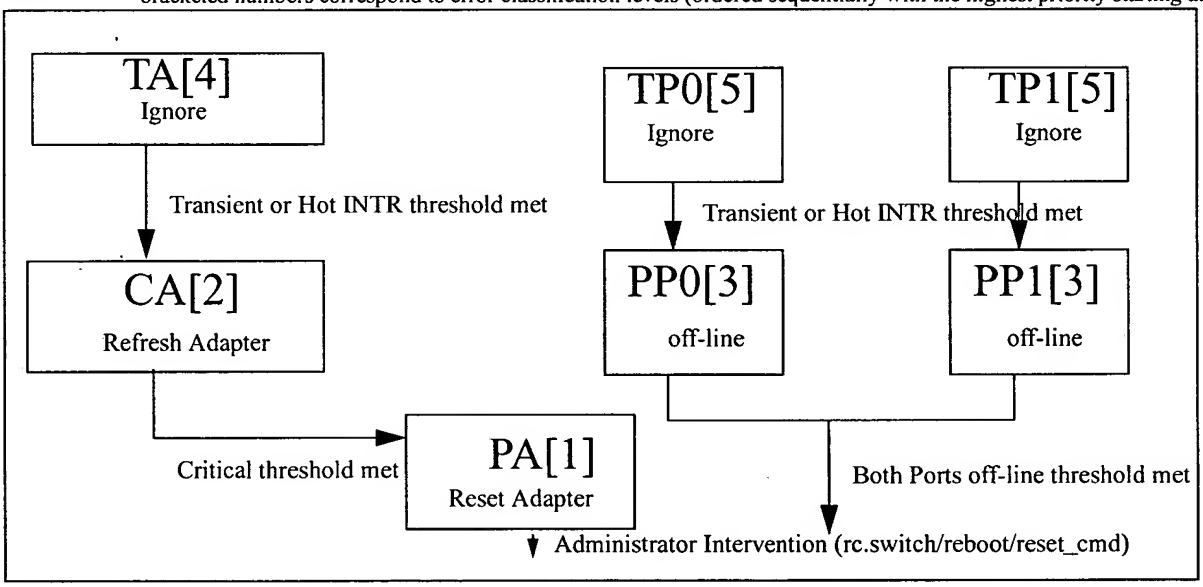
Note: The register and bit description in **bold** face are defined in APPENDIX A for each interrupt register.

3.3 Error Classifications

To determine which hardware and software actions to take, *Col_recovery* defines permanent, critical and transient error classifications at the adapter and port levels. The interrupt status and error registers define a specific error classification for each bit (bit classifications for each error bit are listed in APPENDIX A). The five different error classifications are:

- PA: permanent adapter - Cannot be assigned to a bit (ONLY escalated to).
- CA: critical adapter
- PP0 PP1: permanent port and number
- TA: transient adapter
- TP0 TP1: transient port and number

EXAMPLE 1 This diagram demonstrates the main actions and escalation paths *Col_recovery* takes on specific error classifications. The bracketed numbers correspond to error classification levels (ordered sequentially with the highest priority starting at 1):



The following sections describe the decoding, threshold testing and hardware /software actions associated with each error classification.

3.3.1 Decoding Errors

Col_recovery will call the subroutine *cs_decode_bits()* to decode all active error bits on all adapter error and status registers. *cs_decode_bits()* will:

- return the address of the interrupt error or status register containing the highest level error classification and the bit number that originated the error. If more then one error bit is active at the same classification level the first one found will be returned.

- threshold test all active error bits incrementing thresholds when critical and transient thresholds have not been met. When thresholds have been met *cs_decode_bits()* will:
  - reset the threshold count.
  - escalate the error bit's error thresholds to the next error classification level (pertains to TA thresholds ONLY where dual thresholds are possible).

  Designer Note: *There are only two transient adapter errors on the Colony adapter chips. The new MIC chip has one of these transient adapter errors which is an ECC error that may occur on any one of the four memory processor units. Each unit has a bit with a specific threshold of 64 that may be setup to activate a 6XX interrupt when the threshold has been met. Initially the ECC error bits associated with each MPU will be setup to activate the 6XX interrupt line. Threshold tuning will be done later to determine if some factor of the 64 threshold value is reasonable to use on one or all of the units. The TBIC3 has the only other transient adapter error. Since the MIC transient adapter error may ultimately be setup as a critical adapter error, it is not clear that this error class is truly necessary.*

  - remove the error bit from the current error classification level mask and add it to the escalated error classification level mask (pertains to TA/TP0/TP1 thresholds ONLY).
  - reset all register class masks and TA threshold (may be dual) when returning a PA error. PA errors reset the adapter and await a daemon restart (rc.switch/reboot) or the reset command. The former response will automatically reset these values. However the reset command will issue a reset request to the FSD node queue which will in turn invoke the ADAPTER_START ioctl and restart the adapter and port threads.
- search and threshold test error bits in a top down error classification escalation order as depicted in example 1 (transient adapter escalation search followed by transient port escalation search).

  Designer Note: *Since all errors have the ability to escalate to PA errors, all errors must be searched to determine the highest error classification. For example a TA could escalate to a CA with a threshold of 1 which would threshold to a PA error. TP errors can also do a double threshold from TP to PA when the TP threshold has been met and the other port is already off-line or also in error.*

3.3.1.1 High Level Control Flow

This pseudo code represents the main functional flow of *cs_decode_bits()*:

```
#Include "fsd_multi_thd.h"

Adapter_ErrReg* cs_decode_bits(fs_request *fs_req) {

enum error_regs { NBA_IER, NBA_ISR, MIC_IER, MIC_ISR, TBIC3_IER }
enun error_regs Reg

Adapter_ErrReg   *final_error_reg /* contains final_error_bit & final_error_class */

int hot_intr_threshold = 7
    hot_intr_threshold_interval = 35
    hot_intr_threshold_cnt = 0

/* Read Error registers for ALL transient TA/TP0/TP1 HOT interrupt testing. */
Read NBA_ISR_REG, Read MIC_ISR_REG, Read TBIC3_IER_REG

/* Transient Adapter Errors */
for Reg = MIC_ISR to TBIC3_IER
   for each active error_bit in fs_req.ErrReg[Reg] && Adapter_ErrReg[Reg].TA_mask
      If ( ++error_bit.threshold_cnt == error_bit.threshold (within error_bit.threshold.limit) )
         || ( HOT_INTR_THRESHOLD(error_bit.bit_active_mask, Reg, NBA_ISR, MIC_ISR_REG, TBIC3_IER_REG) )
         error_bit.threshold_cnt = 0
         error_bit.threshold = error_bit.escalted_threshold /* new CA threshold & interval */
         error_bit.threshold_interval = error_bit.escalted_threshold_interval
         remove the error bit from the Adapter_ErrReg[Reg].TA_mask XORED bit_active_mask
         add the error bit to the Adapter_ErrReg[Reg].CA_mask ANDED bit_active_mask
         Adapter_ErrReg[Reg].final_error_bit = 1st escalated TA error_bit#
         Adapter_ErrReg[Reg].final_error_class = CA
         final_error_reg = Adapter_ErrReg[Reg]
      endif
      else
         if ( TA > Adapter_ErrReg[Reg].final_error_class )
            Adapter_ErrReg[Reg].final_error_bit = 1st TA error_bit#
            Adapter_ErrReg[Reg].final_error_class = TA
            final_error_reg = Adapter_ErrReg[Reg]
         endif
   endfor
endfor

/* Critical Adapter Errors */
for Reg =  NBA_IER to TBIC3_IER
   for each active error_bit in fs_req.ErrReg[Reg] && Adapter_ErrReg[Reg].CA_mask
      If ( ++error_bit.threshold_cnt == error_bit.threshold (within error_bit.threshold.limit) )
         error_bit.threshold_cnt = 0
         Adapter_ErrReg[Reg].final_error_bit = 1st escalated CA error_bit#
         Adapter_ErrReg[Reg].final_error_class = PA
         final_error_reg = Adapter_ErrReg[Reg]
         /* Don't search and update any further. Daemon must be restarted resetting all port, and error bit classes and masks.*/
         reset_classmasks_TAthresholds() /* in Adapter_ErrReg struct */
         return  /* on 1st PA error */
      endif
      else
         if ( CA  > AdapterErrReg[Reg].final_error_class ) /* escalated TA-CA error takes precedence */
            Adapter_ErrReg[Reg].final_error_bit = 1st CA error_bit#
```

```
            Adapter_ErrReg[Reg].final_error_class = CA
            final_error_reg = Adapter_ErrReg[Reg]
         endif
      endelse
   endfor
endfor

/* Transient Port Errors */
for Reg = TBIC3_IER
   for each error_bit in fs_req.ErrReg[Reg] && (Adapter_ErrReg[Reg].TP0_mask || TP1_mask)
      If ( ++error_bit.threshold_cnt == error_bit.threshold (within error_bit.threshold.limit) )
         || ( HOT_INTR_THRESHOLD(error_bit.bit_active_mask, Reg, NBA_ISR, MIC_ISR_REG, TBIC3_IER_REG) )
         error_bit.threshold_cnt = 0
         remove the error bit from the Adapter_ErrReg[Reg].TP#_mask XORED bit_active_mask
         add the error bit to the Adapter_ErrReg[Reg].PP#_mask ANDED bit_active_mask
         if ( PP#  > Adapter_ErrReg[Reg].final_error_class ) /* CA error takes precedence */
            Adapter_ErrReg[Reg].final_error_bit = 1st escalated TP# error_bit#
            Adapter_ErrReg[Reg].final_error_class = PP#
            final_error_reg = Adapter_ErrReg[Reg]
         endif
      endif
      else
         if ( TP# > Adapter_ErrReg[Reg].final_error_class ) /* CA/TA errors take precedence */
            Adapter_ErrReg[Reg].final_error_bit = 1st TP# error_bit#
            Adapter_ErrReg[Reg].final_error_class = TP#
            final_error_reg = Adapter_ErrReg[Reg]
         endif
   endfor
endfor

/* Port Permanent Errors */
for Reg = TBIC3_IER
   for each error_bit in fs_req.ErrReg[Reg] && (Adapter_ErrReg[Reg].PP0_mask || PP1_mask)
      PP#_offline = T
      If ( PP0_offline==T  && PP1_offline==T )
         Adapter_ErrReg[Reg].final_error_bit = 1st escalated PP# error_bit#
         Adapter_ErrReg[Reg]..final_error_class = PA
         final_error_reg = Adapter_ErrReg[Reg]
         /* Don't search and update any further. Daemon must be restarted resetting all port and error bit classes and masks.*/
         return on 1st PA error/
      endif
      else
         if ( PP#  > Adapter_ErrReg[Reg].final_error_class ) /* CA error takes precedence */
            Adapter_ErrReg[Reg].final_error_bit = 1st PP# error_bit#
            Adapter_ErrReg[Reg].final_error_class = PP#
            final_error_reg = Adapter_ErrReg[Reg]
         endif
   endfor
endfor

return final_error_reg

} /* cs_decode_bits */
```

3.3.2 Threshold Testing

There are two types of threshold tests. The first being a transient or critical error threshold test and the second a transient error hot interrupt test. Transient bad packets errors are threshold tested in a slightly different manner from the rest of the transient errors.

3.3.2.1 Transient Bad Packet Threshold

TB3.recovey currently measures bad packet thresholds by the frequency in which they occur in relationship to each other within a specific interval of 13.7 seconds. Seven bad packets occurring in less then 13.7 seconds within a specific interval will exceed the threshold. When the threshold is met the error is escalated to a permanent error and the FSD exits.

Col_recovery will perform this threshold test when a bad packet transient port error is raised by the microcode using the same method used by *TB3.recovey.* Since the Colony adapter bandwidth has increased (bad packets will come at a faster rate (x packets/secs)) *Col_recovery* will attempt to tune the interval (bad pings in 1 second time frame). When the threshold is met the port in error will be taken off-line.

3.3.2.2 Transient & Critical Threshold

TB3.recovery currently measures TBIC2 transient error thresholds by the frequency in which they occur in relationship to each other, but not within a specific time interval. All active transient errors are counted as one error case. Here 7 error cases occurring in less then 10 seconds will exceed the threshold. When the threshold is met the error is escalated to a permanent error and the FSD exits.

Col_recovery will perform this threshold test on all transient and critical errors (excluding the bad packet error) and will maintain a threshold for each individual bit on all the adapter interrupt error and status registers on the NBA, MIC and TBIC3 chips.Thus where *TB3.recovery* thresholds on 7 transient error cases within the 10 second interval, *Col_recovery* will only threshold on the same transient error bit within the same 10 second interval. Threshold tuning will be done later to determine the best thresholds for each error. When the threshold is met these errors will be escalated to either refreshing or resetting the adapter or taking the port off-line.

3.3.2.3 Transient Hot Interrupt Threshold

TB3.recovery currently tests TBIC3 transient errors for hot interrupts when the transient error threshold has not been met. *TB3.recovery* reads and clears the TBIC2 error register then and tests to detect if any active transient error bits remain on. If any bits are active the adapter is refreshed. This scenario is reiterated up to 6 times in an attempt to clear all errors. When the errors do not clear the error is escalated to a permanent error and the FSD exits.

Col_recovery will perform this threshold test on all hardware transient errors at the bit level. Each bit initially active when passed from the SLIH will be tested individually. Bits that have become active after the SLIH has read them will be ignored. Since critical adapter errors refresh the adapter (clearing the interrupt error registers) without thresholding this test is not necessary. Microcode generated errors cannot be tested because these errors are never disabled for interrupts by the SLIH. Resetting

the ISR bit before the SLIH can handle it would result in a lost interrupt. Since hardware transients are disabled by the SLIH and re-enabled back by *Col_recovery* this is not a problem. When the threshold is met these errors will be escalated to either refreshing the adapter or taking the port off-line.

3.3.3 Hardware & Software Actions

3.3.3.1 Permanent Adapter

A permanent adapter error is an unrecoverable adapter error that resets the adapter and waits for administrator intervention. Error bits cannot be defined as permanent adapter errors. Permanent adapter error can only be achieved through an escalation path (e.g. critical adapter threshold or both ports off-line).

- Device driver FLIH/Error SLIH will take the appropriate actions associated with the initial error classification before escalation.
- *Col_recovery* will take the following actions:
    - notify all window owners to release their resources without reading or writing SRAM or RAMBUS.
    - reset the adapter if the SLIH has not already done so.
    - terminate the HAL session.
    - add an entry in the AIX error log and take a full snap at the adapter level.
    - shutdown port and adapter threads.
    - wait for the administrator to use rc.switch, reboot or issue the reset command.

3.3.3.2 Critical Adapter

A critical adapter error is a recoverable adapter error where *Col_recovery* attempts to recover the adapter and maintain the running job.

- Adapter hardware has activated the kill interface. DMA master operations are stopped and the ports are fenced. Does not stop DMA slave operations or mastering of interrupts.
- Device driver FLIH/Error SLIH will take the following actions:
    - stopped global interrupts. The NBA will not master any more interrupts. No interrupt error or status registers have been reset or disabled for interrupts.
    - place requests to quiesce all 6XX slave traffic (no reads/writes to SRAM).
    - reset the adapter when 6XX slave traffic has been quiesced within a given time.
    - update request with copy of error registers (all non enabled bits cleared) and place on the ARWQ.

- *Col_recovery* will take the following actions:
  - request that 6XX slave traffic quiesce for errors escalated to CA.
  - verify that 6XX slave traffic has quiesced and reset the adapter for errors escalated to CA or CA errors that have not successfully been reset by the error SLIH.
  - resign as primary/secondary node.
  - AIX error log and take full snap at the adapter level.
  - re-initialize the adapter.
  - resume quiesced 6XX slave traffic.

3.3.3.3 Permanent Port

A permanent port error is an unrecoverable port error that takes the port in error off-line. Administrator intervention is required to recover the port in error.

- Microcode will take the following actions:
  - disable bit in error and all bits associated with the port in error for 6OX interrupts. If both ports were in error all port permanent bits will be disabled for 6OX interrupts.
  - go into a suspended state (only svc packets are processed).
- Device driver FLIH/Error SLIH will take the following actions:
  - disable all bits associated with the port in error for 6XX interrupts. If both ports were in error all port permanent bits will be disabled for 6XX interrupts.
  - disable and reset all bits in error with the exceptions of microcode errors which are not disabled and port errors which are not reset.
  - update request with copy of error registers (all non enabled bits cleared) and place on the ARWQ.
  - disable the port in error.
- *Col_recovery* will take the following actions:
  - suspend the microcode and take the port off-line for errors that have escalated to PP0 or PP1.
  - merge route tables, load route tables and reset the CM (double port support)
  - resume the microcode (allow protocol data packets to resume).
  - AIX error log.
  - shutdown the port thread in error.

3.3.3.4 Transient Adapter and Port

A transient error is ignored and does not effect normal processing.

- Device driver FLIH/Error SLIH will take the following actions:
  - reset and disable all bits in error. Acknowledged microcode on microcode generated errors.
- *Col_recovery* will take the following actions:
  - AIX error log.
  - re-enable 6XX interrupts on all hardware transient errors that have not escalated

3.3.4 Main Functional Flow

This pseudo code represents the main functional flow of *Col_recovery*:

```
#Include "fsd_multi_thd.h"
Adapter_ErrReg   *error_reg_ptr
int  fs_daemon_fsm_main()
while (1)
   switch1 ( request.type )
      case ( hal_pkt_rcvd )
      case ( local_error )
      Begin Col_recovery
         /* Decode bits and threshold test (Colony adapter specific)*/
         error_reg_ptr = cs_decode_bits(fs_request *request)
            case ( TA || TP0 || TP1 )
               write COLA reset reg: disable GLOBAL 6XX STOP INTERRUPTS bit
               cs_write_css_errorlog(error_reg_ptr)
               enable active hardware transient bits for interrupts
            case ( PP0 || PP1 )
               acquire adapter lock(mode=shutdown) on PP#
               if escalted_error
                  /* 6XX intentional 6OX interrupt. Microcode suspends if  not      */
                  /* already (all PP errors) & flushes cache to SRAM (for logging). */
                  ADAPTER_HARD_SUSPEND(cssfd) ioctl /* disables ALL PP# and TP# INTs */
               endif
               if !(PORT_DISABLE(PP#))
                  LOAD_ROUTES(cssfd) ioctl
                  merge route tables
               endif
               do until (errno==MICROCODE_IN_SUSPEND_&_RESUMED || timer_pop)
                  ioctl_rc=ADAPTER_RESUME(cssfd)
               enddo
               if ioctl_rc
                  goto PERM_FAIL
               cs_write_css_errorlog(error_reg_ptr)
               css.snap -P css_num -f
               drain PP# vWRQ and empty vFIFO
               release the adapter work lock on PP#
               call port_mgmt_thread_shutdown(PP#)
               wait until port thread "state" is OFF
               break
```

```
case  ( CA )
   acquire adapter lock(mode=shutdown) on both ports
   If (node == Primary || backup)
      resign personality
   If (escalated_error)
      ADAPTER_WINDOW_SUSPEND(cssfd)
   If (fs_request.reset == F) || (escalated_error)
      do until (!ioctl || timer_pop)
         ioctl_rc=ADAPTER_WINDOW_SUSPEND_COMPLETE(cssfd)
      enddo
      if ioctl_rc
         goto PERM_FAIL
      do until (errno!=EBUSY || timer_pop)
         ioctl_rc=ADAPTER_RESET(cssfd)   /* disables ALL INTs */
      enddo
      if ioctl_rc
         goto PERM_FAIL
   endif
   cs_write_css_errorlog(error_reg_ptr)
   css.snap -A -f css_num -n /* Don't dump cache to SRAM*/
   do until (errno!=EBUSY || timer_pop)
      ioctl_rc=ADAPTER_START(cssfd)   /* enables ALL INTs */
   enddo
   if ioctl_rc
      goto PERM_FAIL
   ADAPTER_RESUME_WINDOW(cssfd)
   release the adapter lock on both ports
   break
case  ( PA )
PERM_FAIL:
   acquire adapter lock(mode=shutdown) both ports
   if ioctl_rc
      cs_write_fsdaemon_prtfile(errno)
   ADAPTER_RESORCE_RELEASE(cssfd, adapter_access=no) /* Close all windows */
   If (fs_request.reset == F)
      do until (errno!=EBUSY || timer_pop)
         ioctl_rc=ADAPTER_RESET(cssfd)   /* disables ALL INTs */
      enddo
      if ioctl_rc
         cs_write_fsdaemon_prtfile(errno)
   endif
   terminate HAL session
   cs_write_css_errorlog(error_reg_ptr)
   If (dump cache to SRAM complete)
      css.snap -A -f css_num -n /* microcode is halted.Don't dump cache to SRAM*/
```

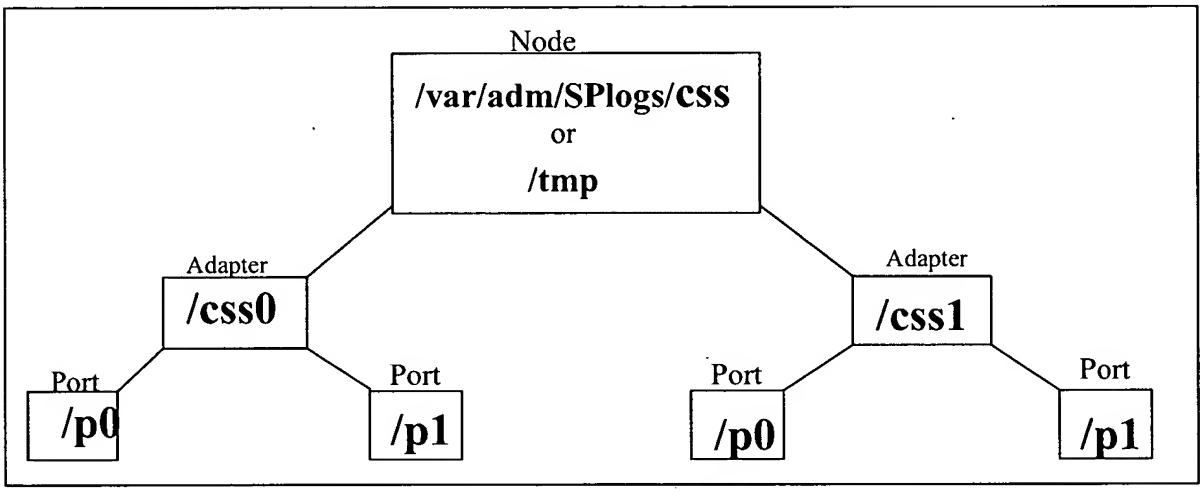
```
                drain both vWRQs and empty vFIFOs
                release the adapter work lock both ports
                call port_mgmt_thread_shutdown(PP0, PP1)
                wait until both port thread "state" are OFF
                drain adapter WRQ
                cleanup local thread memory and call mutex destructors
                return()  /* exits adapter main thread */

          endswitch2
       end Col_recovery
       case ( hal_pkt_rcvd )
endswitch1
end /* fs_daemon_fsm_main */

cs_wite_fsdaemon_prtfile()
   if (errno==ENOTREADY)
      then write HW error msg
      else write SW error msg
end /* cs_write_fsdaemon_prtfile */
```

3.4 Error Logging

3.4.1 Error Logging Directory Tree

The new Colony css error log tree structure is required to eliminate log file access contention between the FSD threads which manage the node, adapter and ports. To achieve this, a new directory level will be created for each adapter and port. The css directory now becomes the node level directory. The snap command must also maintain this hardware hierarchy when creating temporary files.

Planned Directory Structure:



3.4.1.1 Log File Placement

No logs will be shared between multiple threads and some logs have been consolidated into others. Date feild on sanp tar file will be expanded to include seconds.

Merged Log Files

- daemon.stderr and daemon.stdout have been merged into daemon.log
- css_dump.out and fs_dump.out (temporary snap files) have been merged into cad_dump.out
- worm.trace has been merged into fs_daemon_print_file

Obsolete Log Files

- Eclock.log
- xlinx_file

Node Level Files

- core
- css.snap.log
- daemon.log
- Ecommands.log
- logevnt.out
- rc.switch.log
- *date*.node.css.snap.tar: consists of all of the above and the following temporary files:
    - cad_dump.out /* device driver/kernel extension dump of msg buffer*/

- col_dump.out (both adapters)
- errpt.out
- netstat.out (both adapters)
- odm.out (both adapters)
- regs.out (both adapters)
- spdata.out
- ifcl_dump. (both adapters) /* IP dump */
- fifo_dump (both adapters) /* us space and service FIFO dump Conditional flag pssed to css.snap */

Adapter Level Files

- dtbx.trace
- dtbx_failed.trace
- css.snap.log
- scan_out.log
- scan_save.log
- *date*.css#.css.snap.tar: consists of all of the above and the following temporary files:
  - cad_dump.out
  - col_dump.out (adapter specific)
  - errpt.out
  - netstat.out (adapter specific)
  - odm.out (adapter specific)
  - regs.out (adapter specific)
  - spdata.out
  - ifcl_dump (adapter specific)
  - fifo_dump (adapters) specific

Port LevelFiles

- cable.miswire
- css.snap.log
- flt
- fs_daemon_print.file
- out.top
- router.log
- topology.data
- *date*.port#.css.snap.tar: consists of all of the above and the following temporary files:
  - cad_dump.out
  - errpt.out
  - spdata.out

The following rules will be applied to the above snap tar files:

- all tar files will include all pertinent files from the top down. In other words adapter level files will include port levels files. No redundant files will be included.
- safe dumps will be done first.
- don't support NLS.

3.4.2 css.snap Command

The purpose of *css.snap* is to generate a compressed archive of the CSS log files.The *css.snap* command must be expanded to support the new error logging directory structure.

3.4.2.1 Modifications

Synopsis

css.snap [-c | -n] [-s] [-N | -A[css0 | css1 | blank for both] | -P [p0 | p1 | blank for both]]

Options

[-c | -n] [-s] options - behavior unchanged

[-N | -A[css0 | css1 | blank for both] | -P [p0 | p1 | blank for both]]: specifies the hardware level (node/adapter/port) at which the css.snap.log and compressed achieve files are to be generated. Node level is the default when no options are specified. Both files will be placed in the directory level corresponding to the specified hardware level. [-f] specifies user space and service FIFO dump.

Changes

The new snap will do a query to determine which snap to do (old TB3 or new)Colony. The changes for the new Colony snap are:

1. add new hardware level option and parameter checking.
2. add new dump of user space and service FIFO option and parameter checking.
3. include only the permanent and temporary files dictated by the hardware option and new directory tree structure. Snap and archive files will be placed in the directory level corresponding to the specified option.
4. modify command invocations which need to pass the specific adapter.
5. remove tb2 and tb3 logic. This includes removing obsolete and merged files.

3.4.3 AIX Error Logging

*Col*_recovery will not exploit the new error logging facilities for the Colony release due to timing constraints on the Colony release.

3.4.3.1 Message Structure

The following message structure is the currently used structure. It is shown here as the general template that applies to all new error labels unless otherwise listed. The detailed data has been updated to include the new Colony adapter registers, a string identifying which register was in error and a definition of the specific error bit. The bit definitions are listed in APPENDIX A for each interrupt register under error definition.

Currently used error logging message structure:

```
LABEL:
    comment         = ""
    class           = H
    Log             = True
    Report          = True
    Alert           = False
    Err_Type        = PERM
    Err_Desc        = codept to "Same as the comment string"
    Prob_Causes     = codept to "Same as the comment string"
    User_Causes     = codepts to "TBD"
    User_Actions    = codepts to "TBD"
    Fail_Causes     = codepts to "TBD"
    Fail_Actions    = codepts to "TBD"
    Detail_Data     = 80, codept to specific register error is reported on, ALPHA
    Detail_Data     = 80, codept to definition of specific error bit, ALPHA
    Detail_Data     = 16, codept to NBA_ISR, ALPHA
    Detail_Data     = 16, codept to MIC_ISR, ALPHA
    Detail_Data     = 16, codept to MIC_IER, ALPHA
    Detail_Data     = 16, codept to TBIC3_IER, ALPHA
```

3.4.3.1.1 New Error Labels

The new error labels are based on the specific error source that caused them. The new labels are listed in APPENDIX A for each interrupt register under error source.

The following new message labels and their corresponding code points will be added to the CSS error logging template and code point files (*h_erecv_templ* and *ssp.css.codeponit.S)*:

Permanent Adapter

- COL_ADAPT_HW_ER:

```
comment         = "Switch adapter hardware error"
```

- COL_ADAPT_MC_ER:

```
comment         = "Switch adapter microcode error"
```

- COL_ADAPT_SFW_ER:
  ```
  comment        = "Switch adapter software error"
  ```

- COL_ADAPT_HW_MC_ER:
  ```
  comment        = "Switch adapter hardware or microcode error"
  ```

- COL_ADAPT_HW_SFW_ER
  ```
  comment        = "Switch adapter hardawre or software error"
  ```

Permanent Port

- COL_PORT_HW_ER
  ```
  comment        = "Switch adapter port hardware error"
  ```

Transient Adapter

- COL_ADAPT_HW_RE
  ```
  comment        = "Switch adapter transient hardware error"
  Err_Type       = TEMP
  ```

Transient Port

- COL_PORT_HW_RE
  ```
  comment        = "Switch adapter port transient hardware error"
  Err_Type       = TEMP
  ```

- COL_PORT_MC_RE (bad packet)
  ```
  comment        = "Switch adapter port transient hardware error"
  Err_Type       = TEMP
  ```

  Note: The rest of the fields will remain as they are today with the addition of the port number in the interrupt vector structure.

3.4.3.2 Consolidated Error Logging

All new messages will be enabled to participate with CSS consolidated error logging. For each new message label a new ODM error notification object will be added to the *errlog.rm.add* file. These objects will all specify the consolidated error logging method *logevnt*. This method is the event management resource monitor front end for consolidated error logging.

4.0 Threshold Tuning

4.1 Scenarios

TBD

4.2 Results

TBD

APPENDIX A Interrupt Registers Classifications & Setup

The interrupt register tables that follow were derived from the corresponding tables defined in the Scalable Parallel (SP) Colony Communications Adapter Functional Specification. New Columns were added to these tables to define *Col_recovery's* specific classifications and setup requirements.

Table Definitions

Error Status Class : Classification of hardware error or status.

- CA - Critical adapter error followed by 1st guess thresholds
- TA - Transient adapter error followed by 1st guess thresholds
- PP0 PP1 - Permanent port error
- TP0 TP1 - Transient port error followed by threshold
- STAT - Status
- UNEXP - Unexpected status
- CKSTP/ATTN - A checkstop and or attention

Error Status Source: Source that caused the hardware error or status.

- SW740 - Microcode
- SW6k - RS6K software
- HW - Hardware
- HW_SW740 - Hardware or microcode
- HW_SW6k - Hardware or RS6K software
- REC - *Col_recovery*
- default - don't care

6XX INT : Causes an interrupt to the RS6K device driver SLIH.

6OX INT : Causes an interrupt to the adapter microcode on the NBA and TBIC3 chips. Activates the hang indictor bit in the NBA LHS IER for the MIC chip.

Chkstp Attn : Causes a hardware node checkstop or service processor attention on the NBA LHS IER.

Kill INTF : Kill interface line. Fences the 6XX bus and both links.

- Y - Enabled when bit is active
- default - Not enabled

Reset Bit : Software responsible for clearing the error bit.

- DD - RS6k device driver SLIH
- MC - Microcode Interrupt handler
- REC - *Col_recovery*
- SP - Service Processor
- default - don't care

Error or Status Label : Label associated with error/status.

Cause of active bit : The specific cause that activated the error/status bit.

Hardware response : The hardware response to the detected error/status.

***Col_recovery* response**: *Col_recovery* response to the detected error class.

- REFRESH/T-RESET - CA error class

  A critical adapter error will refresh the adapter (reset/re-initialize) when the threshold has not been meet and log the error. If the threshold has been meet the error will escalate to a permanent adapter error which will reset the adapter, log the error and wait for user intervention.

- IGNORE/T-REFRESH - TA error class

  A transient adapter error will log the error and take no recovery actions when the threshold has not been meet. If the threshold has been meet the error will escalate to a critical adapter error which will refresh (reset/re-initialize) the adapter and log the error.

- OFFLINE - PP0/PP1 error classs

  Permanent port error will take the port off-line and log the error if the remaining port is on-line. If the remaining port is already off-line or shares a concurrent port permanent error the error will escalate to a permanent adapter error which will reset the adapter, log the error and wait for user intervention.

- IGNORE/T-OFFLINE - TP0/TP1 error classs

  A transient port error will log the error and take no recovery actions when the threshold has not been meet. If the threshold has been meet the error will escalate to a permanent port error which will take the port off-line and log the error if the remaining port is on-line. If the remaining port is already off-line or shares a concurrent port permanent error the error will escalate to a permanent adapter error which will reset the adapter, log the error and wait for user intervention.

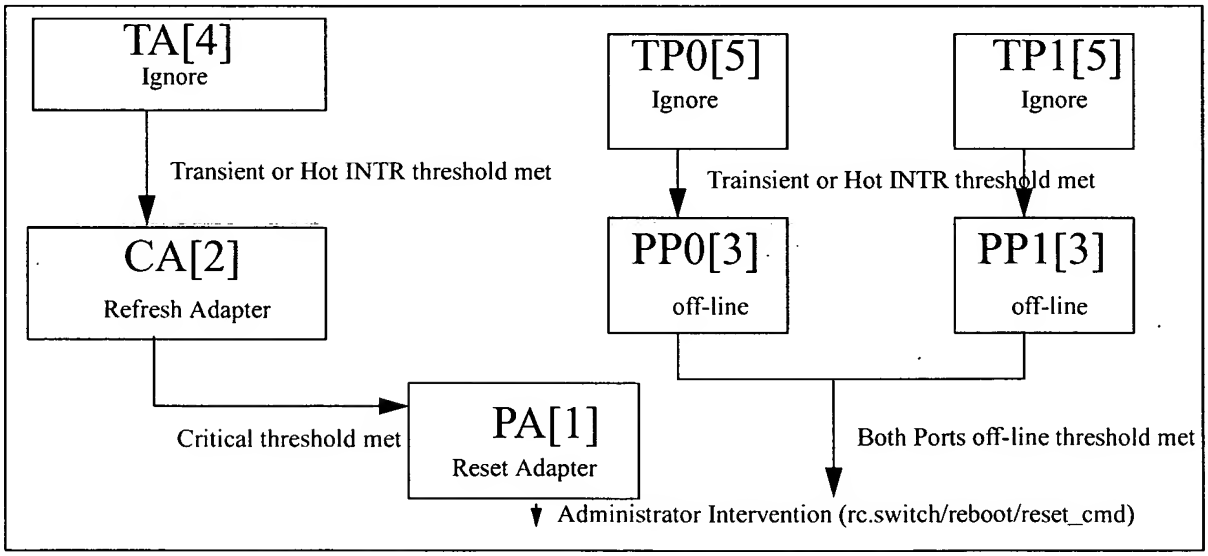
- Investigate the code - Currently TBIC3 CA error class

  Dump bitscan (100k). Two classs full chip and ?. Talk to Greg Saylor about feasibility.

- default - STAT/UNEXP status classs

  Col_recovery does not handle status interrupts.

This diagram demonstrates the main actions and escalation paths *Col_recovery* takes on specific error classifications. The bracketed numbers correspond to error classification levels (ordered sequentially with the highest priority starting at 1):



NBA

RHS Interrupt Status

Address : 2 B0 01 18
Type : Read-XORWrite
JTAG : F1 || F2 r/wxor
Power up Value : 0x0000 0000 0000 0000
: 0xFFFE BFFE 3F00 0084 (6XX Interrupt Enable Register - 2 B0 01 30 R/W)
: 0x0000 0000 0000 000A (6OX Interrupt Enable Register - 2 B0 01 28 R/W)
: 0xFFFE BFFE 3F00 0800 (Kill Interface Enable Register - 2 B0 01 20 R/W)

This register contains the various status indicator bits for the NBA Right Hand Side. When a specific event occurs it will be detected and activate a bit in this register. This detected bit will remain on even if the detected condition stops. This register will NOT be cleared upon reset. It must be written with the same pattern as those bits desired to be reset. The written doubleword is bitwise XOR'd with the contents in the register at the time of the write. Any update to this register will only flag an interrupt signal off-chip if the corresponding enable bit(s) are set in one of the status interrupt enable registers.t

Bit	Error Status Class	Error Status Source	6XX INT	6OX INT	Kill INTF	Reset Bit	Error or Status Label • Cause of active bit • Hardware response • *Col_recovery* response
MSB 0	CA 2-24 hrs	SW740	Y		Y	DD	**DMA boundary error** • A DMA request crossed the 4k boundary. • Read requests will be discarded. Write requests will be discarded, but afterwards the write data buffer will not be able to process the write data correctly. • REFRESH/T-RESET
1	CA 2-24 hrs	HW	Y		Y	DD	**Invalid control flit** • Control flit not marked valid. • Error is logged. Subsequent requests coming in the IB may not be processed correctly. • REFRESH/T-RESET
2	CA 2-24 hrs	HW	Y		Y	DD	**Request to MRRD from PE exceeded** • PE request to MRRD exceeds 256 bytes. • Length of data pushed may be invalid. • REFRESH/T-RESET
3	CA 2-24 hrs	SW740	Y		Y	DD	**Read to MRRD address exceeded** • Read to MRRD address 2B00040 exceeds 8 bytes. • Only 8 bytes of data will be returned. • REFRESH/T-RESET
4	CA 2-24 hrs	HW	Y		Y	DD	**Request to MRRD exceeded** • Request to MRRD exceeds 256 bytes. • None. Data may be corrupted, • REFRESH/T-RESET
5	CA 2-24 hrs	HW	Y		Y	DD	**Request to read MRRD wrapped** • Length of MRRD read request wrapped past zero. • None. Data may be corrupted. • REFRESH/T-RESET
6	CA 2-24 hrs	SW740	Y		Y	DD	**PE push request overflow** • Request received when PE push engine queue is full. • Data may be lost. • REFRESH/T-RESET

Bit	Error Status Class	Error Status Source	6XX INT	6OX INT	Kill INTF	Reset Bit	Error or Status Label • Cause of active bit • Hardware response • *Col_recovery* response
7	CA 2-24 hrs	SW740	Y		Y	DD	**Push request to PE contains invalid address** • PE push request contains invalid destination address. • Error is logged, otherwise, normal operation • invalid destination bit position • REFRESH/T-RESET
8	CA 2-24 hrs	HW	Y		Y	DD	**Invalid IB request** • Selected request bits are not a valid decode. • State machine may be set to zero - this is not normal operation and subsequent operations may be corrupted. • REFRESH/T-RESET
9	CA 2-24 hrs	HW SW740	Y		Y	DD	**Request write to MDD invalid** • Write request to MMD before control register was valid.Data was received before control information. • None. Normal operation not guaranteed. • REFRESH/T-RESET
10	CA 2-24 hrs	SW740	Y		Y	DD	**Request to read empty MDD** • Read request to empty MMD. • None • IGNORE/T-REFRESH
11	CA 2-24 hrs	HW	Y		Y	DD	**Illegal slave request** • slave requests is both EIEIO and Store. • None. Data may be corrupted. • REFRESH/T-RESET
12	CA 2-24 hrs	SW6k Both	Y		Y	DD	**Multiple write requests to NOB_STATUS** • Multiple sources tried to write to the NOB_STATUS register during the same cycle. • None. Data may be corrupted. • REFRESH/T-RESET
13	CA 2-24 hrs	HW SW740	Y		Y	DD	**IB request contains invalid address** • Incoming IB request contains a bad address. • Bad address is save in 2B00228 register. • REFRESH/T-RESET
14	CA 2-24 hrs	SW6k Both	Y		Y	DD	**Concurrent write requests to CRF** • 6XX and 740 tried to write to CRF concurrently. • None. Proper operation is not guaranteed. • REFRESH/T-RESET
15	UNEXP						**Unused**
16	CA 2-24 hrs	HW	Y		Y	DD	**Invalid control flit** • Control flit not marked valid. • None. Subsequent operations may be corrupted. • REFRESH/T-RESET
17	UNEXP						**Unused**
18	CA 2-24 hrs	SW740	Y		Y	DD	**Concurrent MRRD read and PE push** • An explicit read of MRRD address 0x2B00040 and a PE push were attempted concurrently. • None • REFRESH/T-RESET
19	CA 2-24 hrs	HW	Y		Y	DD	**MRRD read address wrapped** • Read address into MRRD unexpected wrap past zero. • None. Data may be corrupted. • REFRESH/T-RESET
20	CA 2-24 hrs	HW	Y		Y	DD	**MRRD running sum error** • Valid bytes in the MRRD buffer exceeds 4KB. • None. Data may be corrupted. • REFRESH/T-RESET
21	CA 2-24 hrs	HW	Y		Y	DD	**MRRD bucket sum error** • Valid buckets in the MRRD exceeds 64. • None • REFRESH/T-RESET

Bit	Error Status Class	Error Status Source	6XX INT	6OX INT	Kill INTF	Reset Bit	Error or Status Label • Cause of active bit • Hardware response • *Col_recovery* response
22	CA 2-24 hrs	HW	Y		Y	DD	**MRRD read push buffer error** • A MRRD read was attempted when neither push buffer was available • None • REFRESH/T-RESET
23	CA 2-24 hrs	HW	Y		Y	DD	**Slave request invalid length** • NBA LHS slave request had an invalid length. • None. Data returned for request may not be correct. • REFRESH/T-RESET
24	CA 2-24 hrs	SW740	Y		Y	DD	**Request to write control FIFO overflow** • Write requests to Control Request FIFO exceeds 16. • None • Request may be lost. Subsequent operation may not be correct. • REFRESH/T-RESET
25	CA 2-24 hrs	HW	Y		Y	DD	**Request to write full MDD** • Write request to full MMD. • Data may be lost. • REFRESH/T-RESET
26	CA 2-24 hrs	HW SW740	Y		Y	DD	**MRRD read request length exceeded** • MRRD read request length exceeded the number of bytes left in the bucket. A push of two non-contiguous chunks of data was attempted. • Data returned may be incorrect. • REFRESH/T-RESET
27	CA 2-24 hrs	HW	Y		Y	DD	**Invalid slave request** • LHS make slave entry 'available' to RHS when valid bit are 00. • None • REFRESH/T-RESET
28	CA 2-24 hrs	SW740	Y		Y	DD	**Concurrent slave read and push request** • An explicit read of MRRD occurred during a NBA push operation. • Read request may not be completed. • REFRESH/T-RESET
29	CA 2-24 hrs	SW740	Y		Y	DD	**Slave read or EIEIO request exceeded** • Outstanding slave read or EIEIO request exceeds two. • None. Normal operation not guaranteed. • REFRESH/T-RESET
30	CA 2-24 hrs	HW	Y		Y	DD	**MDD bucket count error** • MDD bucket count is negative or wrapped to zero. • None. Adapter may hang. • REFRESH/T-RESET
31-33	UNEXP						**Unused**
34-39	CA 2-24 hrs	HW	Y		Y	DD	**IB Macro errors** • An IB macro error occurred. • None • REFRESH/T-RESET
40-43	UNEXP						**Unused**
44	UNEXP						**LHS TRACE START**
45	UNEXP						**LHS TRACE DONE**
46	UNEXP						**RHS TRACE START**
47	UNEXP						**RHS TRACE DONE**
48-51	UNEXP						**Unused**

Bit	Error Status Class	Error Status Source	6XX INT	6OX INT	Kill INTF	Reset Bit	Error or Status Label • Cause of active bit • Hardware response • *Col_recovery* response
52	STAT				Y		KILL INTERFACE activated bit
53	STAT					DD	LHS caused 6xx interrupt bit
54	STAT					DD	LHS received SOFTSTOP bit
55	STAT					DD	LHS received or caused CHECKSTOP bit
56	STAT		Y			DD	LHS reports completed DMA store.*** Effective PASS2 and above ***
57-58	UNEXP						Unused
59	STAT					DD	60X interrupt was activated by any Colony ASIC. *** Effective PASS2 and above ***
60	PP0/1	REC		Y		SW740	6XX causes 60X interrupt • *Col_recovery* interrupt to microcode to hard suspend on escalated transient port error. The microcode will interrogate which port is in error to disable interrupts. • None • OFFLINE
61	TP0/1 7-13.7 secs	SW740	Y			DD	60X causes 6XX interrupt • TP- bad svc/data packet interrupt or service window full from microcode. *Col_recovery* interrogates the bad packet in the INTR vector to determine which port is in error. Must maintain 2 transient thresholds for 1 bit. • None • IGNORE/T-REFRESH
62	STAT			Y		SW740	JTAG causes 60X interrupt
63	STAT					SP	60X causes Service Processor ATTENTION

LHS Interrupt Error

Address : 2 201 A00
Type : Read/XOR-Write
Power up Value : 0x0000 0000 0000 0000 /* Can ONLY be wriiten by SP or COLA reset */
: 0xFFFF FFFF F000 006A (Checkstop Enable Register - 2201C00 R/W)
: 0x0000 0000 0000 0004 (6XX Interrupt Enable Register - 2201D00 R/W)

The Accumulator part of this register is written by exclusive ORing the data from a write command to the bits already in the register.

Bit	Error Status Class	Error Status Source	6XX INT	Chkstp Attn		Reset Bit	Error or Status Label • Cause of active bit • Hardware response • *Col_recovery* response
MSB 0	CKSTP			Y			Parity Error on Master Read Return Data Controls
1	CKSTP			Y			Parity Error on Master Read Return Data
2	CKSTP			Y			Parity Error on Slave Write Data Controls
3	CKSTP			Y			Parity Error on Slave Write Data
4	CKSTP			Y			NBA Data Parity Error detected by sink chip
5	CKSTP			Y			Parity Error in Master Data Control FIFO
6	CKSTP			Y			NBA sourcing bad parity onto 6XX Data bus
7	CKSTP			Y			Non NBA Source chip detected off-bus Error (i.e. Source activated DERR line)
8	CKSTP			Y			Multi-beat Config access attempted
9	CKSTP			Y			Any invalid ARESP as per the Slave Operations table
10	CKSTP			Y			Same as above except access was to LHS config reg
11	CKSTP			Y			ASTATIN Parity Error during any non-null operation
12	CKSTP			Y			ARESPIN Parity Error during any non-null operation
13	CKSTP			Y			ASTATIN was Null on Interrupt Master Op
14	CKSTP			Y			ASTATIN was PosAck on Interrupt Slave Op
15	CKSTP			Y			ASTATOUT was PosAck but ASTATIN was Null
16	CKSTP			Y			ASTATIN was PosAck on EIEIO or SYNC op
17	CKSTP			Y			ASTATIN was PosAck or Retry on Slave Rerun Op
18	CKSTP			Y			Master ASTATIN was ParErr
19	CKSTP			Y			ASTATOUT was Retry but ASTATIN was Null or PosAck

Bit	Error Status Class	Error Status Source	6XX INT	Chkstp Attn		Reset Bit	Error or Status Label • Cause of active bit • Hardware response • *Col_recovery* response
20	CKSTP			Y			ASTATIN was Null on non-SYNC Master Op from NBA
21	CKSTP			Y			Invalid ARESPIN for Master Non-Reads as per Master Operations table
22	CKSTP			Y			ASTATIN was PosAck for Master SYNC
23	CKSTP			Y			NBA Master operations Timed Out
24	CKSTP			Y			Slave Decoded Address Parity Error
25	CKSTP			Y			Extra Data Bus Grant given to NBA
26	CKSTP			Y			Slave Address to NBA was bad. No Colony facility responded.
27	CKSTP			Y			MDD and MDCF out of sync.
28	CKSTP			Y			A Colony PLL lost lock.
29	CKSTP			Y			No address bus grant when requested
30	CKSTP			Y			No data bus grant when requested
31	CKSTP			Y			DCACHE data arrived on not modified read
32-55	UNEXP						Unused
56	STAT						NBA received Checkstop from the Node
57	ATTN			Y			Colony has completed an IB operation from JTAG
58	ATTN			Y			Colony's 60X processor want's SP attention Note>>Enables for this bit should NOT be set active until after the following sequence: RHS's ACCUMULATOR gets reset followed by a reset of this bit (or this register).
59	ATTN						Colony has signalled "KILL_INTF indicator Note>>Enable only after Colony card is fully initialized.
60	STAT			Y		DD	DMA and HW both active.
61	CA 2-24 hrs	HW BOTH by bit	Y			DD	MIC hang condition idicator • MIC hang bit. MIC ISR bits 8,9,10 or 11 is active. High potenial hot interrupt. • None • REFRESH/T-RESET Note>>Enable only after MIC is enabled.
62	ATTN			Y			SELF TEST COMPLETE on a Colony ASIC
63	STAT						NBA turned on Checkstop First.

MIC

Interrupt Status

Address : 3 00 00 10
Type : Read/WriteXOR
JTAG : 04/05 r/wxor
Power up Value : 0x0000 0000 0000 0000
: 0x0000 E0E0 8080 8080 (6XX Interrupt Enable Register - 3 00 00 18 R/W)
: 0x00F0 0000 0000 0000 (6OX Interrupt Enable Register - 3 00 00 20 R/W)
: 0x00F0 E0E0 0000 0000 (Kill Interface Enable Register - 3 00 00 28 R/W)

This register contains the various status indicator bits for the MIC. When a specific event occurs it will be detected and activate a bit in this register. This detected bit will remain on even if the detected condition stops. This register will NOT be cleared upon reset. It must be written with the same pattern as those bits desired to be reset. The written doubleword is bitwise XOR'd with the contents in the register at the time of the write. Any update to this register will only flag an interrupt signal off-chip if the corresponding enable bit(s) are set in one of the status interrupt enable registers.

Bit	Error Status Class	Error Status Source	6XX INT	6OX INT	KILL INTF	Reset Bit	Error or Status Label • Cause of active bit • Hardware response • *Col_recovery* response
MSB 0	UNEXP						**Trace array triggered** • The trigger conditions for the trace array have been met. • Filtered data will begin to be written into the array, and clocks may be stopped.
1	UNEXP						**Unused**
2	UNEXP						**Trace array write** • The trigger and filter have been satisfied. • The array will actually be written
3	UNEXP						**Trace array done** • The array now has 64 valid entries (active until all data and qualifiers are readout) • No new data will be written, and clocks may be stopped.
4-5	UNEXP						**Unused**
6	UNEXP						**Trace bad access** • An update of a control reg during an active trace or a collision between 6XX/60X and JTAG accesses • Operation will continue with resulting controls
7	UNEXP						**Trace array parity error** • A parity error occurred while reading out the trace array.
8	CA 2-24 hrs	HW SW6K	can't access causes hang	Y sets bit LHS NBA	Y LHS NBA can't set		**NBA to MIC Timeout** • Data has been stuck at the IB interface for more than 2^^24 cycles (~0.13 sec) • Operations between chips have stopped or deadlocked • REFRESH/T-RESET

Bit	Error Status Class	Error Status Source	6XX INT	6OX INT	KILL INTF	Reset Bit	Error or Status Label • Cause of active bit • Hardware response • *Col_recovery* response
9	CA 2-24 hrs	HW SW740	can't access causes hang	Y sets bit LHS NBA	Y LHS NBA can't set		**MIC to NBA Timeout** • Data has been stuck at the IB interface for more than 2^^24 cycles (~0.13 sec) • Operations between chips have stopped or deadlocked • REFRESH/T-RESET
10	CA 2-24 hrs	HW SW740	can't access causes hang	Y sets bit LHS NBA	Y LHS NBA can't set		**TBIC to MIC Timeout** • Data has been stuck at the IB interface for more than 2^^24 cycles (~0.13 sec) • Operations between chips have stopped or deadlocked. • REFRESH/T-RESET
11	CA 2-24 hrs	HW SW6K	can't access causes hang	Y sets bit LHS NBA	Y LHS NBA can't set		**MIC to TBIC Timeout** • Data has been stuck at the IB interface for more than 2^^24 cycles (~0.13 sec) • Operations between chips have stopped or deadlocked. • REFRESH/T-RESET
12	UNEXP						**JTAG bad access** • An internal control error occurred such as a bad state mach sequence, fifo overflow or underflow, or bad seq# management.
13	UNEXP						**JTAG command too close** • The MBU detected a bad command or command sequence in a PUSH engine or a bad IB control flit • The MBU will halt?
14-15	UNEXP						**Unused**
16 Dup Errbit 26	CA 2-24 hrs	SW740	Y		Y	DD	**Push Queue 0 bad sequence** • Wrote an address followed by an address or a control followed by a control. • Push queue will halt • REFRESH/T-RESET
17 Dup Errbit 26	CA 2-24 hrs	SW740	Y		Y	DD	**Push Queue 0 bad command** • Invalid combination of 'flag' or 'code' bits in control register. • Push Queue will halt • REFRESH/T-RESET
18 Dup Errbit 26	CA 2-24 hrs	SW740	Y		Y	DD	**Push Queue 0 overflow** • Too many push requests written to the push queue. • The requests have not been lost but this is a warning that writes may back up and cause performance problems. • Read Push Queue status to track available space in queue. • REFRESH/T-RESET
19-23	UNEXP						**Unused**
24 Dup Errbit 29	CA 2-24 hrs	SW740	Y		Y	DD	**Push Queue 1 bad sequence** • Wrote an address followed by an address or a control followed by a control. • Push Queue will halt • REFRESH/T-RESET
25 Dup Errbit 29	CA 2-24 hrs	SW740	Y		Y	DD	**Push Queue 1 bad command** • Invalid combination of 'flag' or 'code' bits in control register. • Push Queue will hal • REFRESH/T-RESET

Bit	Error Status Class	Error Status Source	6XX INT	6OX INT	KILL INTF	Reset Bit	Error or Status Label • Cause of active bit • Hardware response • *Col_recovery* response
26 Dup Errbit 29	CA 2-24 hrs	SW740	Y		Y	DD	**Push Queue 1 overflow** • Too many push requests written to the push queue. • The requests have not been lost but this is a warning that writes may back up and cause performance problems • REFRESH/T-RESET
27-31	UNEXP						**Unused**
32	TA 64-24 hrs 2-24 hrs	HW	Y			DD	**MPU 0 memory error corrected** • The MPU detected a single bit error on a RDRAM read and corrected it. • Good data is passed. • IGNORE/T-REFRESH
33	STAT CA threshold met	HW				DD	**MPU 0 ECC threshold reached** • The MPU has corrected 64 ECC errors on this Rambus channel. • Correction still occurs. • IGNORE/T-REFRESH
34	UNEXP						**MPU 0 performance monitoring complete** • A request for performance monitoring on this channel has completed (64K operations have occurred). • The monitor statistics are frozen.
35	STAT					DD	**MPU 0 RDRAM devices enabled** • All the RDRAM devices have been successfully woken up and enabled by the initialization sequence, as indicated by a high value of the Sin input to the MIC chip from the last RDRAM in the Sin-Sout chain on the channel.
36	STAT					DD	**MPU 0 clean memory operations complete** • The series of operations for writing 0's with good ECC into all of rambus memory is complete.
37	STAT					DD	**MPU 0 current control read operations complete** • The requested number of read operations from the current control register space is complete, establishing the optimal current control value.
38-39							**Unused**
40	TA 64-24 hrs 2-24 hrs	HW	Y			DD	**MPU 1 memory error corrected** • The MPU detected a single bit error on a RDRAM read and corrected it. • Good data is passed. • IGNORE/T-REFRESH
41	STAT CA threshold met	HW				DD	**MPU 1 ECC threshold reached** • The MPU has corrected 64 ECC errors on this Rambus channel. • Correction still occurs. • IGNORE/T-REFRESH
42	UNEXP						**MPU 1 performance monitoring complete** • A request for performance monitoring on this channel has completed (64K operations have occurred). • The monitor statistics are frozen.
43	STAT					DD	**MPU 1 RDRAM devices enabled** • All the RDRAM devices have been successfully woken up and enabled by the initialization sequence, as indicated by a high value of the Sin input to the MIC chip from the last RDRAM in the Sin-Sout chain on the channel.
44	STAT					DD	**MPU 1 clean memory operations complete** • The series of operations for writing 0's with good ECC into all of rambus memory is complete.

Bit	Error Status Class	Error Status Source	6XX INT	6OX INT	KILL INTF	Reset Bit	Error or Status Label • Cause of active bit • Hardware response • *Col_recovery* response
45	STAT					DD	**MPU 1 current control read operations complete** • The requested number of read operations from the current control register space is complete, establishing the optimal current control value.
46-47	UNEXP						**Unused**
48	TA 64-24 hrs 2-24 hrs	HW	Y			DD	**MPU 2 memory error corrected** • The MPU detected a single bit error on a RDRAM read and corrected it. • Good data is passed • IGNORE/T-REFRESH
49	STAT CA threshold met	HW				DD	**MPU 2 ECC threshold reached** • The MPU has corrected 64 ECC errors on this Rambus channel. • Correction still occurs. • IGNORE/T-REFRESH
50	UNEXP						**MPU 2 performance monitoring complete** • A request for performance monitoring on this channel has completed (64K operations have occurred). • The monitor statistics are frozen.
51	STAT					DD	**MPU 2 RDRAM devices enabled** • All the RDRAM devices have been successfully woken up and enabled by the initialization sequence, as indicated by a high value of the Sin input to the MIC chip from the last RDRAM in the Sin-Sout chain on the channel.
52	STAT					DD	**MPU 2 clean memory operations complete** • The series of operations for writing 0's with good ECC into all of rambus memory is complete.
53	STAT					DD	**MPU 2 current control read operations complete** • The requested number of read operations from the current control register space is complete, establishing the optimal current control value.
54-55	UNEXP						**Unused**
56	TA 64-24 hrs 2-24 hrs	HW	Y			DD	**MPU 3 memory error corrected** • The MPU detected a single bit error on a RDRAM read and corrected it. • Good data is passed. • IGNORE/T-REFRESH
57	STAT CA threshold met	HW				DD	**MPU 3 ECC threshold reached** • The MPU has corrected 64 ECC errors on this Rambus channel. • Correction still occurs. • IGNORE/T-REFRESH
58	UNEXP						**MPU 3 performance monitoring complete** • A request for performance monitoring on this channel has completed (64K operations have occurred). • The monitor statistics are frozen.
59	STAT					DD	**MPU 3 RDRAM devices enabled** • All the RDRAM devices have been successfully woken up and enabled by the initialization sequence, as indicated by a high value of the Sin input to the MIC chip from the last RDRAM in the Sin-Sout chain on the channel.
60	STAT					DD	**MPU 3 clean memory operations complete** • The series of operations for writing 0's with good ECC into all of rambus memory is complete.
61	STAT					DD	**MPU 3 current control read operations complete** • The requested number of read operations from the current control register space is complete, establishing the optimal current control value.

Bit	Error Status Class	Error Status Source	6XX INT	6OX INT	KILL INTF	Reset Bit	Error or Status Label • Cause of active bit • Hardware response • *Col_recovery* response
62-63	UNEXP						Unused

Interrupt Error

Address : 3 00 00 30
Type : Read/WriteXOR
JTAG : 0C/0D r/wxor
Power up Value : 0x0000 0000 0000 0000
: 0xF7FE F7FF FFFF FFFF (6XX Interrupt Enable Register - 3 00 00 38 R/W)
: 0xFFFF FFDB FFFF FFFF (Kill Interface Enable Register - 3 00 00 48 R/W)

This register contains the various error detection bits for the MIC. When a specific error occurs it will be detected and activate a bit in this register. This detected bit will remain on even if the detected condition stops. This register will NOT be cleared upon reset. It must be written with the same pattern as those bits desired to be reset. The written doubleword is bitwise XOR'd with the contents in the register at the time of the write. Any update to this register will only flag an interrupt signal off-chip if the corresponding enable bit(s) are set in one of the interrupt enable registers.

Bit	Error Status Class	Error Status Source	6XX INT	6OX INT	KILL INT	Reset Bit	Error or Status Label • Cause of active bit • Hardware response • *Col_recovery* response
MSB 0	CA 2-24 hrs	HW	Y		Y	DD	**IB 1 inbound bus data flit parity error** • Data flit being read into the chip had a parity error. • Bad data will be passe • REFRESH/T-RESET
1	CA 2-24 hrs	HW	Y		Y	DD	**IB 1 inbound bus control flit parity error** • Control flit being read into the chip had a parity error. • Bad operation may occur. • REFRESH/T-RESET
2	CA 2-24 hrs	HW	Y		Y	DD	**IB 1 inbound fifo data flit parity error** • Data being read out from inbound fifo had a parity error. • Bad data will be passed. • REFRESH/T-RESET
3	CA 2-24 hrs	HW	Y		Y	DD	**IB 1 inbound fifo control flit parity error** • Data being read out from inbound fifo had a parity error. • Bad operation may occur. • REFRESH/T-RESET
4	CA 2-24 hrs	HW	Y		Y	DD	**IB 1 control error** • An error was detected in the token counting logic such that there are too many tokens flowing. • Bad data may be passed or data may be lost. • REFRESH/T-RESET
5	CA 2-24 hrs	HW	Y		Y	DD	**IB 1 outbound parity error** • Parity error detected on the 8 byte data leaving the MIC • Bad data may be passed. • REFRESH/T-RESET
6	CA 2-24 hrs	HW	Y		Y	DD	**IB 2 inbound bus data flit parity error** • Data flit being read into the chip had a parity error. • Bad data will be passed. • REFRESH/T-RESET
7	CA 2-24 hrs	HW	Y		Y	DD	**IB 2 inbound bus control flit parity error** • Control flit being read into the chip had a parity error. • Bad operation may occur. • REFRESH/T-RESET
8	CA 2-24 hrs	HW	Y		Y	DD	**IB 2 inbound fifo data flit parity error** • Data being read out from inbound fifo had a parity error. • Bad data will be passed. • REFRESH/T-RESET

Bit	Error Status Class	Error Status Source	6XX INT	6OX INT	KILL INT	Reset Bit	Error or Status Label • Cause of active bit • Hardware response • *Col_recovery* response
9	CA 2-24 hrs	HW	Y		Y	DD	**IB 2 inbound fifo control flit parity error** • Data being read out from inbound fifo had a parity error. • Bad operation may occur. • REFRESH/T-RESET
10	CA 2-24 hrs	HW	Y		Y	DD	**IB 2 control error** • An error was detected in the token counting logic such that there are too many tokens flowing. • Bad data may be passed or data may be lost. • REFRESH/T-RESET
11	CA 2-24 hrs	HW	Y		Y	DD	**IB 2 outbound parity error** • Parity error detected on the 8 byte data leaving the MIC • Bad data may be passed. • REFRESH/T-RESET
12	CA 2-24 hrs	HW SW6K	Y		Y	DD	**IBC 1 invalid address** • Decoded an address that is within MIC register space but has an invalid target or was misrouted to MIC. • The operation is not preformed and is passed onto IBC2 (TBIC). • REFRESH/T-RESET
13	CA 2-24 hrs	HW SW6K	Y		Y	DD	**IBC 1 unimplemented address** • Decoded an address that is within MIC memory space but targets memory that is not installed or is a bad operation to a reg. • The operation is not preformed and is passed onto IBC2 (TBIC). • REFRESH/T-RESET
14	CA 2-24 hrs	HW SW6K	Y		Y	DD	**IBC 1 bad access** • Decoded an access that is not on the correct boundary or an operation that is not allowed to that facility (ex. writing a RO reg). • The operation is not preformed and is passed onto IBC2 (TBIC). • REFRESH/T-RESET
15	CA 2-24 hrs	HW	Y		Y	DD	**IBC 1 bad data length / bad sequence** • More or less data came than was specified in the control flit length field, incorrect size for the specified register, or a control or data flit came in out of order. • A bad operation may occur. • REFRESH/T-RESET
16	CA 2-24 hrs	HW SW740	Y		Y	DD	**IBC 2 invalid address** • Decoded an address that is within MIC register space but has an invalid target or was misrouted to MIC. • The operation is not preformed and is passed onto IBC1 (NBA). • REFRESH/T-RESET
17	CA 2-24 hrs	HW SW740	Y		Y	DD	**IBC 2 unimplemented address** • Decoded an address that is within MIC memory space but targets memory that is not installed or is a bad operation to a reg. • The operation is not preformed and is passed onto IBC1 (NBA). • REFRESH/T-RESET
18	CA 2-24 hrs	HW SW740	Y		Y	DD	**IBC 2 bad access** • Decoded an access that is not on the correct boundary or an operation that is not allowed to that facility (ex. writing a RO reg). • The operation is not preformed and is passed onto IBC1 (NBA). • REFRESH/T-RESET
19	CA 2-24 hrs	HW	Y		Y	DD	**IBC 2 bad data length / bad sequence** • More or less data came than was specified in the control flit length field, incorrect size for the specified register, or a control or data flit came in out of order. • A bad operation may occur. • REFRESH/T-RESET

Bit	Error Status Class	Error Status Source	6XX INT	6OX INT	KILL INT	Reset Bit	Error or Status Label • Cause of active bit • Hardware response • *Col_recovery* response
20	CA 2-24 hrs	HW	Y		Y	DD	**MBU RQ 0 Access Engine error** • An internal control error occurred such as a bad state decode, or a bad command coming into or out of the access engine. • Bad operation may occur. • REFRESH/T-RESET
21	CA 2-24 hrs	HW	Y		Y	DD	**MBU RQ 0 Buffer error** • The MBU detected a parity error on a control flit coming out of the fifo, or a fifo overflow or underflow. • Bad operation may occur. • REFRESH/T-RESET
22	CA 2-24 hrs	HW	Y		Y	DD	**MBU RQ 0 data parity error** • Parity check on a data flit coming out of the request fifo. • Bad data may be written to RDRAM or register. • REFRESH/T-RESET
23	CA 2-24 hrs	HW	Y		Y	DD	**MBU RQ 1 Access Engine error** • An internal control error occurred such as a bad state decode, or a bad command coming into or out of the access engine. • Bad operation may occur. • REFRESH/T-RESET
24	CA 2-24 hrs	HW	Y		Y	DD	**MBU RQ 1 Buffer error** • The MBU detected a parity error on a control flit coming out of the fifo, or a fifo overflow or underflow. • Bad operation may occur. • REFRESH/T-RESET
25	CA 2-24 hrs	HW	Y		Y	DD	**MBU RQ 1 data parity error** • Parity check on a data flit coming out of the request fifo. • Bad data may be written to RDRAM or register. • REFRESH/T-RESET
26 Dup Statbits 16,17,18	CA 2-24 hrs	HW SW740				DD	**MBU RP 0 Push Engine error** • An internal control error occurred such as bad seq# management, bad sequence, or bad command coming into or out of the push engine. • Bad operation may occur. • REFRESH/T-RESET
27	CA 2-24 hrs	HW	Y		Y	DD	**MBU RP 0 Buffer error** • The MBU detected bad parity on a control flit coming out of the fifo, or a fifo overflow or underflow. • Bad operation may occur. • REFRESH/T-RESET
28	CA 2-24 hrs	HW	Y		Y	DD	**MBU RP 0 data parity error** • Parity check on a data flit coming out of the response fifo. • Wrong command could execute or bad data may be passed to main memory or network.
29 Dup Statbits 24,25,26	CA 2-24 hrs	HW SW740				DD	**MBU RP 1 Push Engine error** • An internal control error occurred such as bad seq# management, bad sequence, or bad command coming into or out of the push engine, or bad flow-control. • Bad operation may occur. • REFRESH/T-RESET
30	CA 2-24 hrs	HW	Y		Y	DD	**MBU RP 1 Buffer error** • The MBU detected bad parity on a control flit coming out of the fifo, or a fifo overflow or underflow. • Bad operation may occur. • REFRESH/T-RESET
31	CA 2-24 hrs	HW	Y		Y	DD	**MBU RP 1 data parity error** • Parity check on a data flit coming out of the response fifo. • Wrong command could execute or bad data may be passed to main memory or network. • REFRESH/T-RESET

Bit	Error Status Class	Error Status Source	6XX INT	6OX INT	KILL INT	Reset Bit	Error or Status Label • Cause of active bit • Hardware response • *Col_recovery* response
32	CA 2-24 hrs	HW	Y		Y	DD	**MPU 0 Asynchronous interface error** • This is the interface from the MIC clock to the Rambus (SYN) clock. This will be a fifo pointer error. • Bad operation may be performed. • REFRESH/T-RESET
33	CA 2-24 hrs	HW	Y		Y	DD	**MPU 0 Asynchronous interface error** • This is the interface from the Rambus (SYN) clock to the MIC clock. This will be a fifo pointer error. • Bad operation may be performed. • REFRESH/T-RESET
34	CA 2-24 hrs	HW	Y		Y	DD	**MPU 0 uncorrectable memory error** • An uncorrectable error was detected in a read operation out of rambus memory. • Bad, uncorrectable data in the rambus memory. • REFRESH/T-RESET
35	CA 2-24 hrs	HW	Y		Y	DD	**MPU 0 receive fifo parity error** • Parity check on the data read out of the receive buffer. • Bad data may be passed in the receive path of the MPU or bad operation may be performed. • REFRESH/T-RESET
36	CA 2-24 hrs	HW	Y		Y	DD	**MPU 0 transmit fifo parity error** • Parity check on the data at the end of the transmit path in the MPU. • Bad data may be written to the RDRAM's or bad operation may occur. • REFRESH/T-RESET
37	CA 2-24 hrs	HW	Y		Y	DD	**MPU 0 bad command** • A bad command was received from the MBU. • Bad operation may be performed. • REFRESH/T-RESET
38	CA 2-24 hrs	HW	Y		Y	DD	**MPU 0 protocol error** • A protocol violation to or from the Rambus channel was detected. • Bad operation may be performed. • REFRESH/T-RESET
39	CA 2-24 hrs	HW	Y		Y	DD	**MPU 0 control error** • A parity error occurred on a control flit coming out of either the transmit fifo, the receive fifo, or on it's way to the RAC. • Bad operation may be performed. • REFRESH/T-RESET
40	CA 2-24 hrs	HW	Y		Y	DD	**MPU 1 Asynchronous interface error** • This is the interface from the MIC clock to the Rambus (SYN) clock. This will be a fifo pointer error. • Bad operation may be performed. • REFRESH/T-RESET
41	CA 2-24 hrs	HW	Y		Y	DD	**MPU 1 Asynchronous interface error** • This is the interface from the Rambus (SYN) clock to the MIC clock. This will be a fifo pointer error. • Bad operation may be performed. • REFRESH/T-RESET
42	CA 2-24 hrs	HW	Y		Y	DD	**MPU 1 uncorrectable memory error** • An uncorrectable error was detected in a read operation out of rambus memory. • Bad, uncorrectable data in the rambus memory. • REFRESH/T-RESET
43	CA 2-24 hrs	HW	Y		Y	DD	**MPU 1 receive fifo parity error** • Parity check on the data read out of the receive buffer. • Bad data may be passed in the receive path of the MPU or bad operation may be performed. • REFRESH/T-RESET

Bit	Error Status Class	Error Status Source	6XX INT	6OX INT	KILL INT	Reset Bit	Error or Status Label • Cause of active bit • Hardware response • *Col_recovery* response
44	CA 2-24 hrs	HW	Y		Y	DD	**MPU 1 transmit fifo parity error** • Parity check on the data at the end of the transmit path in the MPU. • Bad data may be written to the RDRAM's or bad operation may occur. • REFRESH/T-RESET
45	CA 2-24 hrs	HW	Y		Y	DD	**MPU 1 bad command** • A bad command was received from the MBU. • Bad operation may be performed. • REFRESH/T-RESET
46	CA 2-24 hrs	HW	Y		Y	DD	**MPU 1 protocol error** • A protocol violation to or from the Rambus channel was detected. • Bad operation may be performed. • REFRESH/T-RESET
47	CA 2-24 hrs	HW	Y		Y	DD	**MPU 1 control error** • A parity error occurred on a control flit coming out of either the transmit fifo, the receive fifo, or on it's way to the RAC. • Bad operation may be performed. • REFRESH/T-RESET
48	CA 2-24 hrs	HW	Y		Y	DD	**MPU 2 Asynchronous interface error** • This is the interface from the MIC clock to the Rambus (SYN) clock. This will be a fifo pointer error. • Bad operation may be performed. • REFRESH/T-RESET
49	CA 2-24 hrs	HW	Y		Y	DD	**MPU 2 Asynchronous interface error** • This is the interface from the Rambus (SYN) clock to the MIC clock. This will be a fifo pointer error. • Bad operation may be performed. • REFRESH/T-RESET
50	CA 2-24 hrs	HW	Y		Y	DD	**MPU 2 uncorrectable memory error** • An uncorrectable error was detected in a read operation out of rambus memory. • Bad, uncorrectable data in the rambus memory. • REFRESH/T-RESET
51	CA 2-24 hrs	HW	Y		Y	DD	**MPU 2 receive fifo parity error** • Parity check on the data read out of the receive buffer. • Bad data may be passed in the receive path of the MPU or bad operation may be performed. • REFRESH/T-RESET
52	CA 2-24 hrs	HW	Y		Y	DD	**MPU 2 transmit fifo parity error** • Parity check on the data at the end of the transmit path in the MPU. • Bad data may be written to the RDRAM's or bad operation may occur. • REFRESH/T-RESET
53	CA 2-24 hrs	HW	Y		Y	DD	**MPU 2 bad command** • A bad command was received from the MBU. • Bad operation may be performed. • REFRESH/T-RESET
54	CA 2-24 hrs	HW	Y		Y	DD	**MPU 2 protocol error** • A protocol violation to or from the Rambus channel was detected. • Bad operation may be performed. • REFRESH/T-RESET
55	CA 2-24 hrs	HW	Y		Y	DD	**MPU 2 control error** • A parity error occurred on a control flit coming out of either the transmit fifo, the receive fifo, or on it's way to the RAC. • Bad operation may be performed. • REFRESH/T-RESET

Bit	Error Status Class	Error Status Source	6XX INT	6OX INT	KILL INT	Reset Bit	Error or Status Label • Cause of active bit • Hardware response • *Col_recovery* response
56	CA 2-24 hrs	HW	Y		Y	DD	**MPU 3 Asynchronous interface error** • This is the interface from the MIC clock to the Rambus (SYN) clock. This will be a fifo pointer error. • Bad operation may be performed. • REFRESH/T-RESET
57	CA 2-24 hrs	HW	Y		Y	DD	**MPU 3 Asynchronous interface error** • This is the interface from the Rambus (SYN) clock to the MIC clock. This will be a fifo pointer error. • Bad operation may be performed. • REFRESH/T-RESET
58	CA 2-24 hrs	HW	Y		Y	DD	**MPU 3 uncorrectable memory error** • An uncorrectable error was detected in a read operation out of rambus memory. • Bad, uncorrectable data in the rambus memory. • REFRESH/T-RESET
59	CA 2-24 hrs	HW	Y		Y	DD	**MPU 3 receive fifo parity error** • Parity check on the data read out of the receive buffer. • Bad data may be passed in the receive path of the MPU or bad operation may be performed. • REFRESH/T-RESET
60	CA 2-24 hrs	HW	Y		Y	DD	**MPU 3 transmit fifo parity error** • Parity check on the data at the end of the transmit path in the MPU. • Bad data may be written to the RDRAM's or bad operation may occur. • REFRESH/T-RESET
61	CA 2-24 hrs	HW	Y		Y	DD	**MPU 3 bad command** • A bad command was received from the MBU. • Bad operation may be performed. • REFRESH/T-RESET
62	CA 2-24 hrs	HW	Y		Y	DD	**MPU 3 protocol error** • A protocol violation to or from the Rambus channel was detected. • Bad operation may be performed. • REFRESH/T-RESET
63	CA 2-24 hrs	HW	Y		Y	DD	**MPU 3 control error** • A parity error occurred on a control flit coming out of either the transmit fifo, the receive fifo, or on it's way to the RAC. • Bad operation may be performed. • REFRESH/T-RESET

TBIC3

Interrupt Error

Address : 100 50
Type : Read/WriteXOR
Power up Value : 0x0000 0000 0000 0000
: 0xFFF8 FDFF FBF7 3F73 (6XX Interrupt Enable Register - 00 58 R/W)
: 0xFFF8 FDFF 7004 0040 (Kill Interface Enable Register - 00 60 R/W)

The bits in this register indicate the occurrence of error events in the TBIC, on the 60X bus, and at the link interfaces. When a specific error occurs it will be detected and activate a bit in this register. This detected bit will remain on even if the detected condition stops. This register will NOT be cleared upon reset. It must be written with the same pattern as those bits desired to be reset. The written doubleword is bitwise XOR'd with the contents in the register at the time of the write. Any update to this register will only flag an interrupt signal off-chip if the corresponding enable bit(s) are set in one of the interrupt enable registers.

Bit	Error Status Class	Error Status Source	6XX INT	6OX INT	KILL INTF	Reset Bit	Error or Status Label • Cause of active bit • Hardware response • *Col_recovery* response
MSB 0	CA 2-24 hrs	HW	Y		Y	DD	**Processor address bus parity error** • Incorrect parity was detected on the processor address bus by TBIC3, or by the 60x as indicated by the input signal P0_T0_APE_N$. The transfer may not have involved TBIC3. • If the operation was a write directed at TBIC3, then data may have been written to the wrong address. • REFRESH/T-RESET
1	CA 2-24 hrs	HW	Y		Y	DD	**Processor data bus parity error** • Incorrect parity was detected on the processor bus during a data transfer by TBIC3, or by the 60x as indicated by the input signal P0_T0_DPE_N$. The data transfer may not have involved TBIC3. • If the operation was a write directed at TBIC3, then corrupted data was written. • REFRESH/T-RESET
2	CA 2-24 hrs	HW	Y		Y	DD	**Internal processor interface data parity error on read** • Incorrect parity was detected at the 60x interface on data being read out of the TBIC. • Normal operation. • REFRESH/T-RESET
3	CA 2-24 hrs	HW	Y		Y	DD	**Asynchronous boundary parity error** • Incorrect parity was detected on an internal register value, such as the TOD, as it was passed from the 125 MHz logic to the 90 MHz logic. • Operation continues. • REFRESH/T-RESET
4	CA 2-24 hrs	HW	Y		Y	DD	**Internal Buffer Error (90Mhz clock domain)** • Incorrect parity was detected on either the Master Request Read Counter, the Master Response Write Counter, the Slave 125 Request Write Counter or the Slave 125 Response Address and/or Data. • Operation continues. • REFRESH/T-RESET

Bit	Error Status Class	Error Status Source	6XX INT	6OX INT	KILL INTF	Reset Bit	Error or Status Label • Cause of active bit • Hardware response • *Col_recovery* response
5	CA 2-24 hrs	SW740	Y		Y	DD	**Invalid TBIC address** • The 60x processor issued an address which decoded to TBIC3 address range, but did not correspond to a supported TBIC3 address or was a read operation of a write only address or a write operation to a read only address. • The bus handshaking will occur. • REFRESH/T-RESET. Write out invalid address at address 00 01 F0. Investigate the code.
6	CA 2-24 hrs	SW740	Y		Y	DD	**Invalid adapter address response received** • The 60x processor issued a no TBIC3 read operation those address did not decode to any other adapter address range. • Normal bus handshaking occurs and the offending address is returned in bits 0:31 of the data bus. • REFRESH/T-RESET. Write out invalid address at address 00 01 F0. Investigate the code.
7	CA 2-24 hrs	SW740	Y		Y	DD	**Send buffer overflow** • The TBIC3 60x slave logic received a write to a full send header or data buffer. • The write operation is prevented from occurring. The free counter will remain at 0. • REFRESH/T-RESET. Investigate the code.
8	CA 2-24 hrs	HW	Y		Y	DD	**Processor Checkstop** • This bit is set when the signal P0_T0_CHKSTP_N$ is activated. • Operation continues. • REFRESH/T-RESET
9	CA 2-24 hrs	SW6K	Y		Y	DD	**Invalid 60x Transfer Size from IB Bus** • This bit indicates that a read request received over the IB bus, which required access to the 60x bus, contained a transfer count that was greater than 64 bytes. Note: see Control Register #1 bit 23 to avoid this error. • Zero's are returned on the read. • REFRESH/T-RESET. Investigate the code.
10	CA 2-24 hrs	SW740	Y		Y	DD	**Invalid 60x write to Address 0x01000100** • This bit indicates that a 60x write operation to address 0x01000100 was received before the fetch response had returned for a previous fetch request, or the combination of the fetch address and fetch size would result in an attempt to fetch across a DW boundary. • The write is discarded and operation continues.write operation. • REFRESH/T-RESET. Investigate the code.
11	CA 2-24 hrs	HW	Y		Y	DD	**Unsolicited Address 0x01000100 Response** • This bit indicates that an unexpected slave address 0x01000100 read response was received. • The response is discarded, and operation continues. • REFRESH/T-RESET
12	CA 2-24 hrs	HW	Y		Y	DD	**Unsolicited Slave Response** • This bit indicates that a slave read response was received that did not match the current operation or an outstanding address retry. • The response is discarded, and operation continues. • REFRESH/T-RESET
13:15	UNEXP						**Unused**
16	CA 2-24 hrs	HW	Y		Y	DD	**Parity error on a data flit at the IB Receive fifo input** • A high value indicates that there was a parity error on a data flit at the input of the IB receive fifo. • Operation continues. • REFRESH/T-RESET

Bit	Error Status Class	Error Status Source	6XX INT	6OX INT	KILL INTF	Reset Bit	Error or Status Label • Cause of active bit • Hardware response • *Col_recovery* response
17	CA 2-24 hrs	HW	Y		Y	DD	**Parity error on a control flit at the IB Receive fifo input** • A high value indicates that there was a parity error on a control flit at the input of the IB receive fifo. • Operation continues. • REFRESH/T-RESET
18	CA 2-24 hrs	HW	Y		Y	DD	**Parity error on a data flit at the IB Receive fifo output** • A high value indicates that there was a parity error on a data flit at the output of the IB receive fifo. • Operation continues. • REFRESH/T-RESET
19	CA 2-24 hrs	HW	Y		Y	DD	**Parity error on a control flit at the IB Receive fifo output** • A high value indicates that there was a parity error on a control flit at the output of the IB receive fifo. • Operation continues. • REFRESH/T-RESET
20	CA 2-24 hrs	HW	Y		Y	DD	**IB Bus token overflow** • A high value indicates that valid flits were received by the receive IB macro while the IB array is full, implying that the sending macro somehow acquired more tokens than is legal. • Operation continues. • REFRESH/T-RESET
21	CA 2-24 hrs	HW	Y		Y	DD	**Parity error on IB Chip Output** • A high value indicates that there was a parity error on the flit presented to the IB macro by the chip internals. • Operation continues. • REFRESH/T-RESET
22	UNEXP						**Unused**
23	CA 2-24 hrs	HW SW740	Y		Y	DD	**DMA Queue Write Error** • The processor has written WD1 of the DMA queue without first writing WD0 or a write to WD0 was followed by another write to WD0 (as a disWD or DW). • The DMA engine will be halted. • REFRESH/T-RESET
24	CA 2-24 hrs	SW6K	Y		Y	DD	**Invalid adapter address received** • A transaction was received over the IB bus with an address that does not decode to the TBIC, the SRAM or was a read of a write only address or a write of a read only address. Or the transaction was a 60x EIEIO request or a 6xx EIEIO response. • On reads a special transaction is returned over the IB bus with an indication that an invalid address was received by the TBIC, and the offending address is returned in bits 0:31and 32:63 of all DW returned. • REFRESH/T-RESET. Write out invalid address at address 00 01 F8.
25	CA 2-24 hrs	HW	Y		Y	DD	**Send Buffer underflow** • The protocol engine logic tried to read from a send buffer when there was no entry available. • Operation continues. • REFRESH/T-RESET
26	CA 2-24 hrs	HW SW740	Y		Y	DD	**Receive data buffer underflow** • Processor reading from an empty receive data buffer. • The read operation will complete. The full counter will remain at 0. • REFRESH/T-RESET
27	CA 2-24 hrs	HW	Y		Y	DD	**Receive buffer overflow** • The protocol engine logic tried to write to a receive buffer when there was no space available. • Operation continues. • REFRESH/T-RESET

Bit	Error Status Class	Error Status Source	6XX INT	6OX INT	KILL INTF	Reset Bit	Error or Status Label • Cause of active bit • Hardware response • *Col_recovery* response
28	CA 2-24 hrs	HW SW740	Y		Y	DD	**Reassembly engine & RCAM logic error** • An internal control error was detected within the reassembly logic, or the RCAM logic received a write to a valid entry, or the addition of a frame base address to a packet offset caused the ReAsm's DMA address to wrap, or the addition of the DMA command address and the DMA transfer size caused an overflow. Note if address increment is disabled so is address overflow checking. Also note if the SRAM Size bits (Control Reg1 bits 26:27) are set an address wrap will be detected accordingly for SRAM addresses, but if they are not set an address wrap will only be detected at 0xFFF FFFF and at 0x0FF FFFF. • If a Reassembly error occurs the Reassembly logic is halted, if an RCAM error occurs operation continues. • REFRESH/T-RESET. Investigate the code.
29	CA 2-24 hrs	HW SW740	Y		Y	DD	**DMA engine error** • An internal control error was detected within the DMA logic, or the addition of the address and the transfer size caused the DMA address to wrap. Note if the SRAM Size bits (Control Reg1 bits 26:27) are set an address wrap will be detected accordingly for SRAM addresses, but if they are not set an address wrap will only be detected at 0xFFF FFFF and at 0x0FF FFFF. • The DMA logic is halted. • REFRESH/T-RESET. Investigate the code.
30	CA 2-24 hrs	HW	Y		Y	DD	**Internal Buffer Error (125Mhz clock domain)** • Incorrect parity was detected on data read out of a buffer and/or on a counter and/or under/over flow occurred. • Operation continues. • REFRESH/T-RESET
31	CA 2-24 hrs	SW740	Y		Y	DD	**DMA Queue Overflow** • The 60x processor has tried to write an entry into the DMA queue such that greater than 8 entries are outstanding. • The request to push the queue entry is held within the chip. If too many entries are written, the requests will back up onto the 60x bus and hold up the processor until the DMA queue clears and is able to receive the pending entries. • REFRESH/T-RESET. Investigate the code.
32	TA 7-10 secs 2-24 hrs	HW	Y			DD	**Send Data Buffer parity error** • An internal parity error was detected on data being read out of the send data buffer. • A packet fail character is sent, and the rest of the packet is discarded. • IGNORE/T-REFRESH
33	CA 2-24 hrs	HW	Y		Y	DD	**Protocol engine control error** • A control error was detected by the protocol processing engine logic. This error is caused by a state machine being in an illegal state, a parity error on data being read from the send header buffer, a parity error on a link buffer pointer, or a parity error on a copy of the control registers. • The switch interfaces are disabled. • REFRESH/T-RESET
34	CA 2-24 hrs	SW740	Y		Y	DD	**Illegal header or route code** • An illegal header or route code was read out of the port 0 send header buffer. The conditions to set this bit are: Source Route with '00' in bits 2:3 of the route code; A value of '11' in bits 6:7 of the header flag field; A value of '101' in bits 5:7 of the header flag field, and a value of '0000' in bits 0:3 of the header flag field; a value of 14 or 15 in the Total Route Nibble Count of the route code (bits 4:7 of Route Control Byte 1). • The packet is not sent, and the switch interfaces are disabled. • REFRESH/T-RESET. Investigate the code.

Bit	Error Status Class	Error Status Source	6XX INT	6OX INT	KILL INTF	Reset Bit	Error or Status Label • Cause of active bit • Hardware response • *Col_recovery* response
35	TP0 7-10 secs	HW	Y			DD	**Port 0 crc error** • A CRC error was detected in the received packet on port 0. • If the error was on the header or service, an entry is placed into the receive service buffer and the rest of the packet is discarded. If the error was on the data, the header is returned as normal, with a return code indicating that the error occurred. The handling of the data is determined by bit 17 of Control Register #0. • IGNORE/T-OFFLINE
36	TP0 7-10 secs	HW	Y			DD	**Port 0 sequence error** • An illegal sequence of switch control characters was received on port 0. • If the error occurs before the complete header and header crc has been received, then an entry is placed on the receive service buffer, and the rest of the packet is discarded. Otherwise, the header is returned as normal, with a return code indicating that the error occurred. The handling of the data is determined by bit 17 of Control Register #0. • IGNORE/T-OFFLINE
37	STAT					DD	**Adapter Kill Asserted** • The interface signal P0_KILL_INTF_N$ was asserted either by the TBIC3 or by an external source. • The Switch interfaces are disabled.
38	TP1 7-10 secs	HW	Y			DD	**Port 1 crc error** • A CRC error was detected in the received packet on port 1. • If the error was on the header or service, an entry is placed into the receive service buffer and the rest of the packet is discarded. If the error was on the data, the header is returned as normal, with a return code indicating that the error occurred. The handling of the data is determined by bit 17 of Control Register #0. • IGNORE/T-OFFLINE
39	TP1 7-10 secs	HW	Y			DD	**Port 1 sequence error** • An illegal sequence of switch control characters was received on port 1. • If the error occurs before the complete header and header crc has been received, then an entry is placed on the receive service buffer, and the rest of the packet is discarded. Otherwise, the header is returned as normal, with a return code indicating that the error occurred. The handling of the data is determined by bit 17 of Control Register #0. • IGNORE/T-OFFLINE
40 SETUP	TP0 2-5 mins	HW	Y			DD	**Link 0 EDC error threshold exceeded** • The number of accumulated link EDC errors matched the EDC error threshold (bits 48:55 of **Control Register #1**). • If bit 35 of **Control Reg. #0** is set, the STI link is retimed between packets. • IGNORE/T-OFFLINE
41	TP0 7-10 secs	HW	Y			DD	**Link 0 recv undefined character error** • An unidentified switch control character was detected on the link. • The character is discarded. • IGNORE/T-OFFLINE
42	PP0	HW	Y			DD	**Link 0 recv sync error** • An error was detected in the receive synchronization logic. Could be caused by the clock frequency of the adapter and the switch being spread too far apart. • Operation continues. • OFFLINE unless the link has been fenced.
43	PP0	HW	Y			DD	**Link 0 recv overflow error** • More data was received than there were outstanding tokens for. • A packet fail character is sent into the TBIC logic, and the rest of the packet is discarded. • OFFLINE

Bit	Error Status Class	Error Status Source	6XX INT	6OX INT	KILL INTF	Reset Bit	Error or Status Label • Cause of active bit • Hardware response • *Col_recovery* response
44	STAT					DD	**Link 0 recv sti error** • The receiver was not able to retime the STI link in three attempts. This error is set after every third attempt at retiming the link. • Attempts continue to be made to retime the STI link.
45	CA 2-24 hrs	HW	Y		Y	DD	**Link 0 recv parity error** • A parity error was detected on data being read out of the TBFC receive buffer. • Operation continues. • REFRESH/T-RESET
46 Setup	TP0 5-1 min	HW	Y			DD	**Link 0 recv tag token error threshold exceeded** • The number of accumulated link tag/token errors met the tag/token error threshold (bits 56:59 of **Control Register #1**). • If bit 35 of **Control Reg. #0** is set, the STI link is retimed between packets. • IGNORE/T-OFFLINE
47	TP0 7-10 secs	HW	Y			DD	**Link 0 recv state error** • An init state machine error was detected in the TBFC logic. • The state machine resets itself and tries to retime the link. • IGNORE/T-OFFLINE
48 Ask Peter if there are any actions we should take (TOD invalid not an error)	STAT					DD	**TOD Update Error** • An update was not received for the TOD register from the switch within 8 TOD cycles. The TOD should be updated every TOD cycle, which is 65536 x 13.3ns = 871 us. • Bit 0 of the TOD is cleared. Operation continues. • The TOD master may be down and need to be taken over.
49	STAT					DD	**TOD EDC Error detected on the link** • An uncorrectable error was detected on the TOD flit from the link. • The TOD is not updated with the erroneous TOD.
50	TP0 7-10 secs	HW	Y			DD	**Link 0 send parity error** • A parity error was detected on the data being read out of the TBFC send fifo. • A packet fail character is sent out, and the rest of the packet is ignored. • IGNORE/T-OFFLINE
51	PP0	HW	Y			DD	**Link 0 send token overflow** • More tokens were received than there are believed to be destination receive buffer entries. • Operation continues, with the token maximum adhered to. • OFFLINE
52 Setup	TP1 2-5 mins	HW	Y			DD	**Link 1 EDC error threshold exceeded** • The number of accumulated link EDC errors met the EDC error threshold (bits 48:55 of **Control Register #1**). • If bit 35 of **Control Reg. #0** is set, the STI link is retimed between packets. • IGNORE/T-OFFLINE
53	TP1 7-10 secs	HW	Y			DD	**Link 1 recv undefined character error** • An unidentified switch control character was detected on the link. • The character is discarded. • IGNORE/T-OFFLINE
54	PP1	HW	Y			DD	**Link 1 recv sync error** • An error was detected in the receive synchronization logic. Could be caused by the clock frequency of the adapter and the switch being spread too far apart. • Operation continues. • OFFLINE unless link has been fenced.

Bit	Error Status Class	Error Status Source	6XX INT	6OX INT	KILL INTF	Reset Bit	Error or Status Label • Cause of active bit • Hardware response • *Col_recovery* response
55	PP1	HW	Y			DD	**Link 1 recv overflow error** • More data was received than there were outstanding tokens for. • A packet fail character is sent into the TBIC logic, and the reset of the packet is discarded. • OFFLINE
56	STAT					DD	**Link 1 recv sti error** • The receiver was not able to retime the STI link in three attempts. This error is set after every third attempt at retiming the link. • Attempts continue to be made to retime the STI link.
57	CA 2-24 hrs	HW	Y		Y	DD	**Link 1 recv parity error** • A parity error was detected on data being read out of the TBFC receive buffer. • Operation continues. • REFRESH/T-RESET
58 Setup	TP1 5-1 min	HW	Y			DD	**Link 1 recv tag token error threshold exceeded** • The number of accumulated link tag/token errors met the tag/token error threshold (bits56:59 of **Control Register #1**). • If bit 35 of **Control Reg. #0** is set, the STI link is retimed between packets. • IGNORE/T-OFFLINE
59	TP1 7-10 secs	HW	Y			DD	**Link 1 recv state error** • An init state machine error was detected in the TBFC logic. • The state machine resets itself and tries to retime the link. • IGNORE/T-OFFLINE
60-61	UNEXP						Unused
62	TP1 7-10 secs	HW	Y			DD	**Link 1 send parity error** • A parity error was detected on the data being read out of the TBFC send fifo. • A packet fail character is sent out, and the rest of the packet is ignored. • IGNORE/T-OFFLINE
63	PP1	HW	Y			DD	**Link 1 send token overflow** • More tokens were received than there are believed to be destination receive buffer entries. • Operation continues, with the believed token maximum is adhered to. • OFFLINE

APPENDIX B Setup of IER/ISR Related Registers

NBA

Event Tree

The NBA event tree register controls interrupts, status and stop DMA actions for the colony adapter (see NBA Functional Specification). This tree must be initialized properly at power on time.

The service processor and device driver are expected to initialize the NBA event tree register to 0x0FFC which enables the following:

- bit 4 - RHS 6XX interrupt will enable 6XX interrupt
- bit 5 - MIC/TBIC 6XX interrupt will enable 6XX interrupt
- bit 6 - LHS 6XX interrupt will enable RHS status
- bit 7 - LHS 6XX checkstop will enable RHS status
- bit 8 - 6XX incoming checkstop will enable RHS status
- bit 9 - LHS 6XX interrupts will enable kill interface
- bit 10 - LHS 6XX checkstop will enable kill interface
- bit 11 - Incoming checkstop will enable kill interface
- bit 12 - Kill interface will enable stop DMA
- bit 13 - 6XX softstop will enable stop DMA

TBIC3

Control Register 0

The device driver is expected to enable bit 35 of control register 0 at initialization time. This bit enables automatic STI retimes when the STI requests a retime or the EDC error threshold or the receive tag token threshold has been exceeded.

Control Register 1

The device driver is expected to initialize bits 48 though 59 to 0x033 which will set both the EDC error and receive tag token thresholds to 3. This threshold and *Col_recovery's* transient thresholds for both these errors are setup to be consistent with TBS recovery.

APPENDIX C Propagating Errors

NBA

050498
Re: J. C. Gregerson

LHS Event Tree Reg

When bit 7 is active, i.e. enable BIU_ENABLE_60X_INT_FR_LHS_CHKSTP, then any LHS status-accum bit from 0:55 will cause bit 55 of the RHS status-accum to fire. This is a checkstop case, so you ma not care about it since 6XX recovery code can't run after checkstop.

LHS Accumulator Reg

When bit [58] is set from RHS_STATUS_ACCUMULATOR[63] (called RHS_TO_LHS_CAUSE_ATTENTION) without passing through any masks.

MIC

05/03/98
Re: M. Grassi

Status Reg

bit 7 - will cause other parity errors on MIC and other chips

bit 8-11 will cause bit 61 of NBA ISR (hang bit tied to 6OX line)

all other bits in the status register should have no specific side effects.

Error Reg

may be caused by NBA RHS status bit 39

bit 0 - will cause MIC err bit 2,
can also cause MIC err bit 11, 15, 22, 25, 26, 29, 36, 44, 52, 60
may be caused by NBA RHS status bit 39

bit 1 - will cause MIC err bit 3,
can also cause MIC err bit 4, 11, 12, 13, 14, 15, 20, 21, 23, 24, 36, 37, 44, 45, 51, 52, 60, 61

bit 2 - same as bit 0

bit 3 - same as bit 1

bit 4 - may cause MIC err bit 15,

bit 5 - should cause NBA RHS status reg either bits 34,36 or 35,37

may be caused by TBIC error bit 21

bit 6 - will cause MIC err bit 8,
can also cause MIC err bit 5, 19, 22, 25, 26, 29, 36, 44, 52, 60

may be caused by TBIC error bit 21

bit 7 - will cause MIC err bit 9,
can also cause MIC err bit 5, 10, 16, 17, 18, 19, 20, 21, 23, 24, 36, 37, 44, 45, 51, 52, 60, 61

bit 8 - same as bit 6

bit 9 - same as bit 7

bit 10 - may cause MIC err bit 19,

bit 11 - should cause TBIC error reg either bits 16,18 or 17,19

bit 12 - should cause TBIC error reg bit 24

bit 13 - should cause TBIC error reg bit 24

bit 14 - should cause TBIC error reg bit 24

bit 15 - can also cause MIC err bit 20, 21, 23, 24, 36, 37, 44, 45, 51, 52, 60, 61

bit 16 - should cause NBA RHS Stat reg bit 13

bit 17 - should cause NBA RHS Stat reg bit 13

bit 18 - should cause NBA RHS Stat reg bit 13

bit 19 - can also cause MIC err bit 20, 21, 23, 24, 36, 37, 44, 45, 51, 52, 60, 61

bit 20 - can also cause MIC err bit 37, 45, 52, 61

bit 21 - can also cause MIC err bit 20, 36, 37, 44, 45, 51, 52, 60, 61

bit 22 - can also cause MIC err bit 25, 26, 29, 36, 44, 52, 60

bit 23 - can also cause MIC err bit 37, 45, 52, 61

bit 24 - can also cause MIC err bit 23, 36, 37, 44, 45, 51, 52, 60, 61

bit 25 - can also cause MIC err bit 25, 26, 29, 36, 44, 52, 60

bit 26 - no likely side effect

bit 27 - should cause MIC err bit 5,
also NBA RHS status reg either bits 34,36 or 35,37

bit 28 - should cause MIC err bit 5,
also NBA RHS status reg either bits 34,36 or 35,37

bit 29 - no likely side effect

bit 30 - should cause MIC err bit 11,
also TBIC error reg either bits 16,18 or 17,19

bit 31 - should cause MIC err bit 11,
also TBIC error reg either bits 16,18 or 17,19

bit 32 - no likely side effect

bit 33 - no likely side effect

bit 34 - should cause MIC err bit either 27,28,5
also NBA RHS status reg either bits 34,36 or 35,37
or
should cause MIC err bit either 30,31,11
also TBIC error reg either bits 16,18 or 17,19

bit 35 - should cause MIC err bit either 27,28,5
also NBA RHS status reg either bits 34,36 or 35,37
or
should cause MIC err bit either 30,31,11
also TBIC error reg either bits 16,18 or 17,19

bit 36 - no likely side effect

bit 37 - no likely side effect

bit 38 - no likely side effect

bit 39 - no likely side effect

bit 40 - no likely side effect

bit 41 - no likely side effect

bit 42 - should cause MIC err bit either 27,28,5
also NBA RHS status reg either bits 34,36 or 35,37
or
should cause MIC err bit either 30,31,11
also TBIC error reg either bits 16,18 or 17,19

bit 43 - should cause MIC err bit either 27,28,5
also NBA RHS status reg either bits 34,36 or 35,37
or
should cause MIC err bit either 30,31,11
also TBIC error reg either bits 16,18 or 17,19

bit 44 - no likely side effect

bit 45 - no likely side effect

bit 46 - no likely side effect

bit 47 - no likely side effect

bit 48 - no likely side effect

bit 49 - no likely side effect

bit 50 - should cause MIC err bit either 27,28,5
also NBA RHS status reg either bits 34,36 or 35,37
or
should cause MIC err bit either 30,31,11
also TBIC error reg either bits 16,18 or 17,19

bit 51 - should cause MIC err bit either 27,28,5
also NBA RHS status reg either bits 34,36 or 35,37
or
should cause MIC err bit either 30,31,11
also TBIC error reg either bits 16,18 or 17,19

bit 52 - no likely side effect

bit 53 - no likely side effect

bit 54 - no likely side effect

bit 55 - no likely side effect

bit 56 - no likely side effect

bit 57 - no likely side effect

bit 58 - should cause MIC err bit either 27,28,5
also NBA RHS status reg either bits 34,36 or 35,37
or
also NBA RHS status reg either bits 34,36 or 35,37
or
should cause MIC err bit either 30,31,11
also TBIC error reg either bits 16,18 or 17,19

bit 59 - should cause MIC err bit either 27,28,5
also NBA RHS status reg either bits 34,36 or 35,37
or
should cause MIC err bit either 30,31,11
also TBIC error reg either bits 16,18 or 17,19

bit 60 - no likely side effect

bit 61 - no likely side effect

bit 62 - no likely side effect

bit 63 - no likely side effect

HANG DETECTS

MIC error reg bits:

if 4, 15 or 20 are on don't attempt to access MIC or TBIC
if 10 or 19 are on don't attempt te TBIC but the MIC should be OK
if 32, 33, 38, 39, 40, 41, 46, 47, 48, 49, 54, 55, 56, 57, 62, 63 are on don't read Rambus

MIC status reg bits:

if 8, 9, 10, 11 are on don't attempt to access MIC or TBIC
if 35, 43, 51, or 59 are NOT on then don't read Rambus

TBIC3

05/12/98
Re: P. Szwed

ERROR REG

bit 1 - processor data parity error

bit 3 - async interface parity error

bit 21 - IB parity error on input to send macro from TBIC internals

bit 45 - link 0 parity error on tbfc recv

bit 56 - link 1 parity error on tbfc recv

Designer Note: *A port error should never propagate a critical error. In other word if both are concurrently present the critical adapter error will always take precedence.*

APPENDIX D TB3MX Recovery High Level Control Flow

```
print worm trace(dump IERs/MBA_stat/mxcfg3/ivec/TSR & label)
print css error log

select
   /* label = Hardware Permanent */
   case (P60X_hang || MBA_ERR || IER_HW_ERR || IER2_HW_ERR)
      If (P60X_hang)
         /* Toggle the 60X hold line several times to flush bad addresses and release */
         /* bus hang. 60X hold line keeps processor off 60X bus.                       */
         ToggleBusHold; css.snap -n
      else css.snap
      goto PERM_ERR

   /* label = Microcode Permanent */
   case (IER2_SW_ERR && TSR_STI1_CLK_VALID) || (IER_SW_ERR) || (P603_INT && IVEC_SUICIDE)
      css.snap
      goto PERM_ERR

   /* label = Bad Packet Recoverable */
   case (P603_INT && IVEC_DATABUFF/SVCBUFF/KEY_ERR)
      if (bad_packet_threshold_met) /* limit exceeded with time limit */
         goto PERM_ERR

   /* label = Transient Recoverable */
   case  (IER_TRANS) || (IER2_TRANS && TSR_STI1_CLK_VALID) || (P603_INT &&
      IVEC_SVC_RECV_QFULL)
      if (transient_threshold_met) /* limit exceeded within time limit */
         goto PERM_ERR
      if (hotint) /* read and try to clear IER (adapter_refresh) before giving up. */
         goto PERM_ERR

   /* label = Clock Outage Recoverable */
   case  (IER_STI0_CLKERR) || (IER2_STI1_CLKERR && TSR_STI1_CLK_VALID) ||
      (IER_SEND_SEQ_ERR) || (IER_TRANS && (!TSR_STI0_STI1_CLK_VALID) ||
      (TSR_STI0_REQUEST_RETIME))
      goto CLK_OUTAGE
endselect
EnableTbic
goto SUCCESS

PERM_ERR:  drain_fs_request_queue
           DisableTbicPort
CLK_OUTAGE:Resign as Prim/Sec & cleanup threads
           /* Stops US/IP protocols from using the switch for I/O.              */
           /* Used for permanent or long term outages and Efence.               */
           /* Stops interrupts (no DMA), disconnects recv port from switch...etc.*/
           /* Does not take the microcode off-line.                             */
           stop_protocols
           wTbicTOD(zero)
           if (PERM_ERR)
             exit 137 /* Must rc.switch */
           else break and get next request /* May be able to recover by Unfence */
 SUCCESS:  process DMA
```

```
/* IER2_SW_ER */
#define IER2_SOFTWARE_LOGIC(16,17)            \
        (IER2_SEND_TO_DISABLED_PORT_0       | \
         IER2_SEND_TO_DISABLED_PORT_1)

/* IER2_HW_ERR */
#define IER2_PERMANENT_HARDWARE               \
        (IER2_TBFC1_SEND_DATA_PARITY        | \
         IER2_TBFC1_SEND_FIFO_OVERFLOW      | \
         IER2_TBFC1_SEND_INIT_FSM           | \
         IER2_TBFC1_SEND_FSM                | \
         IER2_TBFC1_RECV_INIT_FSM           | \
         IER2_TBFC1_RECV_FSM                | \
         IER2_TBFC1_RECV_FIFO_PARITY        | \
         IER2_READ_PARITY_AT_SIDEBAND_BUS   | \
         IER2_WRITE_PARITY_AT_SIDEBAND_BUS  | \
         IER2_SYNCH_ERR_PORT_0_TOKEN        | \
         IER2_SYNCH_ERR_PORT_0_DATA_TAG     | \
         IER2_SYNCH_ERR_PORT_1_TOKEN        | \
         IER2_SYNCH_ERR_PORT_1_DATA_TAG)

/* IER2_STI1_CLKERR */
#define IER2_TEMPORARY_OUTAGEIER2             \
        (IER2_STI1_CLOCK)(15)

/* IER2_TRANS */
#define IER2_TRANSIENT_ERROR((1,3),(1,3,4,5,6,7,8))  \
        (IER2_TBFC1_SEND_SEQUENCE           | \
         IER2_TBFC1_SEND_TOKEN_CODE_SEQ     | \
        (IER2_TBFC1_SEND_SEQUENCE           | \
         IER2_TBFC1_SEND_TOKEN_CODE_SEQ     | \
         IER2_TBFC1_SEND_TOKEN_CTR_OVERFLOW | \
         IER2_TBFC1_RECV_EDC                | \
         IER2_TBFC1_RECV_DATA_SEQ           | \
         IER2_TBFC1_RECV_TOKEN_QUERY_BOUNDS | \
         IER2_TBFC1_RECV_DATA_OVERFLOW)

/* TSR */
 #define TSR_WRAP_MODE                      0x80000000
 #define TSR_RECEIVE_PORT_CONNECTED         0x40000000
 #define TSR_SEND_PORT_CONNECTED            0x20000000
 #define TSR_STI0_CLOCK_DETECTED_VALID      0x10000000 /* Same as TSR(1) Port0 conneceted to switch */
 #define TSR_STI_CLOCK_SELECTED             0x08000000
 #define TSR_RECEIVE_STI0_REQUESTING_RETIME 0x04000000
 #define TSR_SEND_STI_REQUESTING_RETIME     0x02000000
 #define TSR_RECEIVE_PORT1_CONNECTED        0x01000000
 #define TSR_SEND_PORT1_CONNECTED           0x00800000
 #define TSR_STI1_CLOCK_DETECTED_VALID      0x00400000 /*Same as TSR(7) Port1 connected to switch */
 #define TSR_STI1_CLOCK_SELECTED            0x00200000
 #define TSR_RECEIVE_STI1_REQUESTING_RETIME 0x00100000
 #define TSR_SEND_STI1_REQUESTING_RETIME    0x00080000
 #define TSR_PROCESSOR_WROTE_SIDEBAND       0x00040000
 #define TSR_EXTERNAL_WROTE_SIDEBAND        0x00020000
```

APPENDIX E Alternative Interrupt Handler Design

The following design demonstrates the interrupt handler with the moving of disabling stop global interrupts to the *Col_recovery.*

Pros:

- no longer need to disable transients errors from the off-level error SLIH and re-enable from *Col_recovery.* This eliminates:
  - locking overhead
  - reading the 6XX interrupt enable registers to determine if the active bit actually caused interrupt.
  - writing the 6XX interrupt enable registers to disable the bit for interrupts.
- no longer need to drain the AWRQ to take fast action. Instead the local error is put at the head of the queue. Since disabling of stop global interrupts is now done from *Col_recovery* ONLY one local error request may be present on the queue at any given time. Thus the local errors are serialized (handled in the order in which they occurred). This eliminates:
  - flooding the AWRQ with local errors which is currently a problem with the TB3 recovery code.

Cons:

- Worse from a performance stand point. Interrupts are restarted from *Col_recovery* versus the off-level error SLIH. On the normal path preceding service and data packets are stalled longer on port permanent and transient errors.
- The switch could backup on transient errors unless the microcode is suspended.
- The microcode could time-out (13 seconds) waiting for an ack back on the next DMA complete or error raised.

Functional Flow

BEGIN DD FLIH or System SLIH

```
interrupt_type = none; microcode_interrupt = F
/*-------------------------------------------------------------------------------------------------------------------------------------------------------------*/
/* No new interrupts will be generated from the NBA after a critical adapter or interchip bus hang error have been handled.                                   */
/*-------------------------------------------------------------------------------------------------------------------------------------------------------------*/
If (interrupt was not from this interrupt handler's device (eg:css0/css1))
   return INTR_FAIL /* Processor will give to next device */
/*-------------------------------------------------------------------------------------------------------------------------------------------------------------*/
/* If a reset is in progress (ADAPTER_RESET ioctl also acquires lock) we should return successfully. Since the reset will disable                             */
/* interrupts the NBA will not regenerate this interrupt. If another source has acquired the lock the interrupt will be regenerated.                          */
/*-------------------------------------------------------------------------------------------------------------------------------------------------------------*/
If (acquire the hard reset interlock) return INTR_SUCC
read NBA_ISR
```

```
/*-------------------------------------------------------------------------------------------------------------------------------------------------------*/
/*- If the KILL interface is active we MUST have a critical adapter error. ONLY critical adapter errors and the                                          */
/*  ADAPTER_RESET ioctl activate the KILL interface and a reset would have kept us from getting the lock.                                                */
/*- Critical adapter errors MUST supersede DMA complete because we should not access SRAM. we cannot guarantee that the                                  */
/*  critical adapter error has not caused a bus hang which will hang the processor when a read to the INTR VECTOR fails to                               */
/*  return. DMA will not be regenerated because CA errors stop interrupts and reset the adapter before we exit the error SLIH.                           */
/* - DMA completes MUST override hardware generated transient and port permanent errors. Determining these errors takes too                             */
/*  many processor cycles to be in the main FLIH path. These errors will not be reset & disabled. Interrupt will be regenerated.                         */
/*  Port permanent error cannot get starved because the microcode is suspended (stops data DMA). Transient error could get starved                       */
/*  but is highly unlikely (we will take the chance).                                                                                                     */
/*-------------------------------------------------------------------------------------------------------------------------------------------------------*/
if (NBA_ISR ANDed NBA_ISR_KILLmask)
   interrupt_type = critical_adapter_error          /* Hardware generated error. Includes interchip bus hangs. */
   ADAPTER_SUSPEND_WINDOW ioctl equivalent                                  /* quiesce 6XX slave traffic HAL/IP */
endif
else
   if (NBA_ISR ANDed NBA_ISR_DMAmask)                                  /* Microcode generated DMA. Never disabled. */
      interrupt_type = microcode_DMA
      read INT_VECTOR from SRAM global window          /* Multiple DMAs (window thresholds may be set)*/
      call appropriate DMA-SLIHs (HAL/KHAL/IP) /* generate events */
      /* reset active DMA complete and expected status bits NOT active error bits */
      write back NBA_ISR reg with ( NBA_ISR ANDed NBA_ISR_DMASTATmask )
      release the hard reset interlock
      micrcode_interrupt = T
   endif
   else if (NBA_ISR ANDed NBA_ISR_740mask) /* Can never have both microcode interrupts */
      interrupt_type = microcode_trans_error                  /* Microcode generated error. Never disabled */
      read INT_VECTOR from SRAM global window /* svc window full or bad packet NOT both */
      tmp_fs_request.intVect = Interrupt Vector;
      tmp_fs_request.err_rc = Bad Packet Description
      tmp_fs_request.err_data = Error data pertaining to bad packet description
      write back NBA_ISR reg with ( NBA_ISR ANDed NBA_ISR_740mask )     /* reset active error bit*/
      microcode_interrupt = T
   endif
   /* If DMA we cannot determine if we also have a TP/TA/PP error without reading regs    */
   if (interrupt_type == "") || (NBA_ISR ANDed NBA_ISR_HWerrormask)
      interrupt_type = hardware_trans_or_port_error                        /* Hardware generated.   */
endelse
/*-------------------------------------------------------------------------------------------------------------------------------------------------------*/
/* Error condition because we have no DMA.                                                                                                               */
/*-------------------------------------------------------------------------------------------------------------------------------------------------------*/
If (interrupt_type != microcode_DMA)
   /*----------------------------------------------------------------------------------------------------------------------------------------------------*/
   /* MUST stop the NBA from mastering new interrupts before they can be disabled. Does not disable interrupt enable register.    */
   /*----------------------------------------------------------------------------------------------------------------------------------------------------*/
   write NBA_COLA reset reg to activate GLOBAL 6XX STOP INTERRUPTS bit
   Call the system to schedule off-level error handler(NBA_IER, NBA_ISR, interrupt_type, tmp_fs_request)
endif
if (microcode_interrupt == T)
   ack back to microcode (write SRAM) so new interrupt can be generated (DMA or error)
return INTR_SUCC
```

END DD FLIH or System SLIH

```
/*-------------------------------------------------------------------------------------------------------------------------------*/
/*                                        BEGIN OFF-LEVEL ERROR SLIH                                                             */
/*-------------------------------------------------------------------------------------------------------------------------------*/
Error_interrupt = F
select interrupt_type
       case (critical_adapter_error)
          /*-------------------------------------------------------------------------------------------------------------------*/
          /* Col_recovery will resume suspended windows. If we can't put on the AWRQ the DD will release resources.          */
          /*-------------------------------------------------------------------------------------------------------------------*/
          do until (errno!=EBUSY || timer_pop)
             ioctl_rc=ADAPTER_WINDOW_SUSPEND_COMPLETE(cssfd) equivalent
          enddo
          if ioctl_rc
             ADAPTER_RESORCE_RELEASE(cssfd, adapter_access=no)
             interrupt_type = permanent_adapter
          endif
          toggle COLA reset reg: activate 740 SOFT RESET bit /* MC dump SRAM wait for hard reset */
          write COLA reset reg: activate 740 HARD RESET bit /* Take MC off-line */
          write COLA reset reg: reset RHS_NBA/MIC/TBIC3/LHS_NBA enable /* reset chips */
          wait 50 bus clocks /*sleep 1 minute */
          write COLA reset reg: disable reset RHS_NBA/MIC/TBIC3/LHS_NBA enable /* out of reset */
       case (hardware_trans_or_port_error || microcode_trans_error) /* may have both */
          /*-------------------------------------------------------------------------------------------------------------------*/
          /* - Since critical adapter errors never disable the global stop interrupts (allowing interrupts to be mastered again) */
          /* we need not disable interrupt enable registers or clear IER/ISR bits. Interrupt enables were disabled through the  */
          /* above chip resets and IER/ISR are cleared through step 14 of the ADAPTER_START ioctl at Col_recovery time.         */
          /* - Update - stop global interrupts are now disabled after Col_recovery is given control and has completed decoding */
          /* bits. Thus transient errors never need be to disabled for interrupts and therefor do not need to read the interrupt */
          /* enable registers. Port permanent errors are the ONLY errors that need to be disabled for interrupts and thereby   */
          /* also need to read the interrupt enable register to ensure that the error actually caused the interrupt.           */
          /*-------------------------------------------------------------------------------------------------------------------*/
          fs_request.ErrReg[0] = NBA_ISR
          if (fs_request.ErrReg[0])
             Error_interrupt = Error_interrupt ANDed NBA_ISR_ACTIVEmask
             if (interrupt_type != critical_adapter || permanent_adapter)
                /* reset active and 6XX enabled ISR bits. Don't want to reset 740 interrupt...MC may have enabled again. */
                write back NBA_ISR with (NBA_ISR ANDed NBA_6XX_ISMR)XORed NBA_ISR_740mask
             endif
          endif
          read MIC_ISR
          fs_request.ErrReg[1] = MIC_ISR
          if (fs_request.ErrReg[1])
             Error_interrupt = Error_interrupt ANDed MIC_ISR_ACTIVEmask
             if (interrupt_type != critical_adapter || permanent_adapter)
                /* reset active and 6XX enables ISR bits. Must obtain new lock e.g. ADAPTER_REGISTER_UPDATE */
                write back MIC_ISR with ( MIC_ISR·ANDed MIC_6XX_ISMR )
                read MIC_ISR /* read the MIC ISR again for Hot interrupt testing.
                fs_request.MIC_ISRsnap = MIC_ISR
             endif
          endif
```

```
            read MIC_IER
            fs_request.ErrReg[2] = MIC_IER ANDed MIC_6XX_IEMR
            if (fs_request.ErrReg[2])
               Error_interrupt = Error_interrupt ANDed MIC_IER_ACTIVEmask
               if (interrupt_type != critical_adapter || permanent_adapter)
                  /* reset active and 6XX enables IER bits */
                  write back MIC_IER with ( MIC_IER ANDed MIC_6XX_IEMR )
               endif
            endif
            read TBIC3_IER
            read TBIC3_6XX_IEMR
            fs_request.ErrReg[3] = TBIC3_IER ANDed TBIC3_6XX_IEMR
            if (fs_request.ErrReg[3])
               Error_interrupt = Error_interrupt ANDed TBIC3_IER_ACTIVEmask
               if (interrupt_type != critical_adapter || permanent_adapter)
                  /* If port permanent error disable interrupts for all port error bits associated with the port in error.  */
                  if TBIC3_IER ANDed (TBIC3_IER_PP0mask || TBIC3_IER_PP1mask)
                     write back TBIC3_6XX_IEMR with ( ~( TBIC3_IER ANDed TBIC3_6XX_IEMR )
                                                ORed TBIC3_IER_PP#TP#mask )
                  /* reset active & 6XX enables IER bits but NOT shared 60X IER bits (microcode needs to see INTR too.  */
                  write back TBIC3_IER with (TBIC3_IER ANDed (TBIC3_6XX_IEMR XORed TBIC3_60Xmask))
                  read TBIC3_IER /* read the TBIC3 IER again for Hot interrupt testing     */
                  fs_request.TBIC3_IERsnap = TBIC3_IER
               endif
            endif
endselect
fs_request.ActiveErr_mask = Error_interrupt
fs_request.interrupt_type = interrupt_type
release the hard reset interlock
if(Error_interrupt)
   call fault_service_inotice(fs_request) /* put the request on the head of the AWAQ */
      /* Adapter thread may be hung */
      do until (put request.ERROR on AWRQ(at head fast request)) || (timer_pop)
      if !(put request.ERROR on AWRQ)
         /* notify protocols, reset adapter, dump the AWRQ and take snap */
         put request.AWRQ_FULL on CWRQ
      endif
   end /* fault_service_inotice*/
endif
/*-------------------------------------------------------------------------------------------------------------------------------*/
/*                          END OFF-LEVEL ERROR SLIH                                                                             */
/*-------------------------------------------------------------------------------------------------------------------------------*/
```